Towards Effective Developer Recommendation in Software Crowdsourcing

Shixiong Zhao, Beijun Shen[*], Yuting Chen, Hao Zhong
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, China
{649707869, bjshen, chenyt, zhonghao}@sjtu.edu.cn

***Abstract*—Crowdsourcing has attracted increasing attention from both industry and academia since it was proposed. Now a lot of work is finished by crowdsourcing, such as logo design, website promotion, industrial design, copywriting, software development, translation and image annotation. Although software crowdsourcing achieves positive results in practice, we still face a challenge of assigning suitable developers to specific tasks. In this paper, we propose a novel approach that recommends developers. In particular, our approach supports: comprehensively measuring the tasks and developers in software crowdsourcing, and recommending developers on the basis of the developer-task competence, task-task similarity, and soft power.**

Keywords- software crowdsourcing; developer recommendation; developer model

I. Introduction

Crowdsourcing is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call [1]. It has attracted great attention from both industry and academia. Now a lot of work is finished by crowdsourcing, such as logo design, website promotion, industrial design, copywriting, software development, translation and image annotation. Meanwhile, when employing crowdsourcing to accomplish software development tasks, we face a challenge of assigning suitable developers to specific tasks. Up to now, most development tasks are assigned in a form of bidding or competition. As a result, much human effort is wasted. Although many attendees would compete for the tasks, some tasks are not well accomplished, since they are not assigned to the most suitable developers.

To the best of our knowledge, most crowdsourcing platforms still rely on crowdsourcers to assign tasks. As a result, crowdsourcers have to spend much time in matching the tasks and the developers. To make things worse, a crowdsourcer is usually biased, since he or she may not have a thoroughly understanding of all developers.

To address the above problem, in this paper, we propose a novel approach that recommends suitable developers to tasks in software crowdsourcing. In particular, our approach uses two models to measure tasks and developers, respectively. For given tasks and developers, our approach computes their developer-task competences, task-task similarities, and soft powers, and recommends the best candidate(s) to each task.

II. Related Work

Most research on crowdsourcing is concentrated on several topics, including how to apply crowdsourcing [2, 3], how to control the quality of crowdsourcing [4, 5], how to allocate crowdsourcing tasks.

Few researchers discuss how to recommend workers. In particular, I. Boutsis and V. Kalogeraki [6] present REACT that schedules tasks for the crowd under time constraints. It collects worker profiles (*e.g.*, real-time computational capabilities) and dynamically assigns tasks to suitable workers. Research [7] advocates replacing “pull” with “push” for task allocation to achieve higher quality. They analyze the social network of the crowd to gain better performance. E. Simpson and S. Roberts [8] present an information-theoretic approach to assigning workers to specific tasks in crowdsourcing using a Bayesian method. Research [9] proposes to use auctions to map tasks to workers. They organize auctions taking into account price and the suitability of workers estimated based on generated user profiles.

Meanwhile, the existing researches mainly focus on simple tasks, *e.g.*, image annotation, and most researchers just do theoretic research or provide their frameworks. In addition, most of their researches are not domain specific and not complete. They are not able to be applied to software crowdsourcing. We provide an approach to recommend suitable developers to software development tasks.

III. Task Model and Developer Model

Our approach first abstracts software development tasks and developers with two models. The models measure the tasks and developers quantitatively and thus help establish matching relations between them.

A. *Task Model*

We model a task as [ID, software category, software size, ability requirement(s), enrolled deadline, task deadline, budget, location].

Each task is associated with one or more ability requirements, and each ability is composed of three sub-attributes: “*type*” and “*name*” are the type and name of an

[*] corresponding author

DOI reference number: 10.18293/SEKE2015-091

TABLE I. DEVELOPER MODEL

Attribute	Sub-attribute	Field
ID	/	/
Base information	Address	/
	Education background	Degree
		Major
	Work experience	Job
		Duration
Service range	/	/
Ability and development experience	Ability	Type
		Name
		Level(1..5)
	Development experience	Date
		Task
		Earning
		Evaluation
Credit and guarantee	Credit	/
	Guarantee	
Task-specific information	Task-ID	
	Bid price	
	Delivery time	

ability requirement; and "*level*" represents the required degree of skills. For example, [*coding language, Java, 3*] indicates that the task requires a Java programmer at a level of "3".

A task has other attributes: "*software category*" is the kind of the software; "*software size*" is the size of the software, and it could be an estimated value given by the crowdsourcer for the developers to estimate a reasonable delivery time; "*enrolled deadline*" is the deadline for the enrollment; and "*task deadline*" is the deadline to complete the task. Each task is also associated with "*budget*", working "*location*".

B. Developer Model

A developer model defines the attributes of developers. As shown in Table I, the developer model has the key attributes of a developer in software crowdsourcing.

"*Base information*" includes personnel information of a developer: "*address*" determines whether a developer can be assigned when a task has the location requirement; "*education background*" and "*work experience*" are strong reference; "*service range*" indicates what kinds of software a developer is able to develop.

The "*ability and development experience*" attribute helps crowdsourcer decide whether or not a developer is competent for a task: "*ability*" is similar to the ability requirement in the task model; "*development experience*" shows the history of a developer in completing software tasks in the crowdsourcing platform.

"*Guarantee*" is the guarantee provided by the developer. It is measured by the security deposit of a developer. "*Credit*" shows the status of development credits: after completing each crowdsourcing task, the developer is evaluated by the crowdsourcer. When calculating the credit of a developer, we consider these factors: if two developers have the same credit value (e.g., four stars), the one with more earnings is preferred (reward-related); the credit changes over time, thus a credit that is gained a long time ago has a weak effect (time-related).

The task-specific information is associated with the task that the developer is enrolled in. It includes task-ID, bid price, and promised delivery time.

IV. DEVELOPER RECOMMENDATION

After a task is released, the crowd can browse and enroll for the task. We recommend the suitable developers among the candidates to tasks. The crowdsourcer selects developers to accomplish tasks according to recommendation list.

Employing our task model and developer model, we calculate the developer-task competence ($com_{d\text{-}t}$), the task-task similarity ($sim_{t\text{-}t}$) and the soft power of developers (*sftpwr*), when recommending developers. In particular, the developer-task competence is used to measure whether a developer is competent for a task; the task-task similarity measures whether a developer has good similar work experience; and soft power reflects personal inner qualities, *e.g.*, the credit of the developer. The three factors complement each other and are useful to match tasks and developers comprehensively. Based on them, we calculate recommendation index (recindex) of a developer by (1). The larger *recindex* is, the higher the developer will be in the recommendation list.

$$recindex = \alpha \times com_{d\text{-}t} + \beta \times sim_{t\text{-}t} + \gamma \times sftpwr \qquad (1)$$
$$\text{where } \alpha + \beta + \gamma = 1.$$

Next, we will introduce how to calculate the developer-task competence, the task-task similarity and the soft power.

A. Developer-task competence

The developer-task competence ($com_{d\text{-}t}$) measures whether a developer satisfies the requirements of a task. We calculates this value by

$$com_{d\text{-}t} = com_{dln} \cdot com_{ctg} \cdot (com_{bgt} + com_{loc} + com_{abi})/3 \qquad (2)$$

It is calculated from the software category competence (com_{ctg}), the ability competence (com_{abi}), the task deadline competence (com_{dln}), the budget competence (com_{bgt}), and location competence (com_{loc}). The requirement of task deadline and category is compulsory, so in (2), com_{dln} and com_{ctg} act as multipliers. The formula of com_{dln}, com_{ctg}, com_{bgt}, com_{loc}, and com_{abi} are defined in Table II.

B. Task-task similarity

If a developer has the development experience of similar tasks in the crowdsourcing platform, it will be an important

TABLE II. CALCULATION OF DEVELOPER-TASK COMPETENCE FACTORS

Factor	Value	Condition
com_{dln}	1	*delivery time ≤ task deadline*
	0	otherwise
com_{bgt}	1	*bid price ≤budget*
	Budget / bid price	otherwise
com_{loc}	1	the developer's address is (in) the required location
	0	otherwise
com_{ctg}	1	*the developer's service range cover the required software category*
	0	otherwise
com_{abi0}	$com_{typ} \cdot com_{nam} \cdot com_{lev}$ calculating one of required abilities of the task	
com_{abi}	$1/n \times \sum_{i=0}^{n} (com_{abi0})_i$ there are n required abilities in the task	
com_{typ}	1	*developer has the ability with the same type of the required ability*
	0	otherwise
com_{nam}	1	*developer has the ability with the same name of the required ability*
	0.5	*developer only has the ability with the name under the required type* e.g., name of required ability: C, name of developer's ability: C++
	0	otherwise
com_{lev}	1	$level_w \le level_r$
	$level_w / level_r$	otherwise

reference. As a result, we take the task-task similarity into account, and here defines a task as T = [*cat*, *typ*, *nam*, *lev*, *bgt*]. The *i-th* done task of a developer is presented with T_i = [cat_i, typ_i, nam_i, lev_i, bgt_i, evl_i], where *Cat*, *typ*, *nam*, *lev* and *bgt* represent category, ability type, ability name, ability level, and budget, repectively; and *evl* is the evaluation of a developer gotten from the crowdsourcer on this task.

We use Soft Jaccard's Coefficient to measure the task-task similarity. Suppose that the evaluation value is an integer from 1 to 5, and the developer has n finished tasks in total, sim_{t-t} is calculated as follows:

$$sim_{t-t} = max\left\{\frac{|T \cap T_i|}{|T \cup T_i|} \cdot \frac{evl_i}{5}\right\}(i = 1, \cdots, n) \quad (3)$$

For *cat*, if *cat* = cat_i, they are regarded as intersected. Otherwise they are disjoint. So are *typ* and *nam*. For *lev*, if the name of the *i-th* finished task is same with the task, and lev_i is not lower than lev, they are regarded as intersected. Otherwise, they are disjoint. For *bgt*, if $1/5 \le bgt/bgt_i \le 5$, they are regarded as intersected. Otherwise, they are disjoint. When judging whether they are intersected or not, the condition is relaxed but not just depending on whether they are the same, so we call it Soft Jaccard's Coefficient.

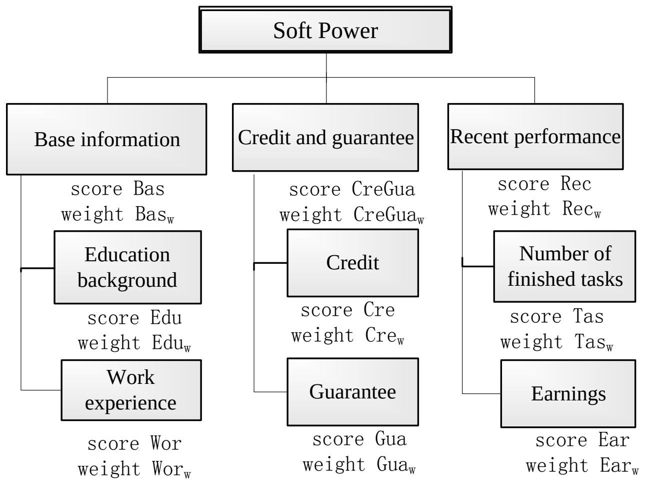

Figure 1. Composition of soft power

Although a developer has experience on a similar task, she/he may not do the task well. Considering this issue, when calculating the task-task similarity, our appoach introduces the crowdsourcer's evaluation, evl_i. All the similar tasks of a developer are calculated one by one, and we select the maximum value as sim_{t-t}.

C. Soft power

Some information of a developer are not reflected in com_{d-t} and sim_{t-t}, *e.g.*, the education background, work experience (not in software crowdsourcing), the credit, the guarantee of a developer, and the recent performance. We call these factors soft power. The composition of soft power is in Fig. 1. "*sftpwr*" is calculated by

$$sftpwr = Bas \times Bas_w + CreGua \times CreGua_w + Rec \times Rec_w \quad (4)$$
$$\text{where } Bas_w + CreGua_w + Rec_w = 1$$

Now we will focus on "*Bas*", "*CreGua*" and "*Rec*".

1) "Bas"

The base information includes education background and work experience. The original "Bas_0" is calculated by (5). Suppose that *BASE* is the maximum value among all the enrolled developers, we divide "Bas_0" by *BASE* to get "*Bas*".

$$Bas_0 = Edu \times Edu_w + Wor \times Wor_w \quad (5)$$
$$\text{where } Edu_w + Wor_w = 1$$

The education background includes some [degree, major] tuples. "*Edu*" is calculated by (6). The work experience includes some [job, duration] tuples. "*Wor*" is calculated by (7). The score of degree (*Degree*) is 1, 2, 3 for bachelor, master, doctor respectively, and 0 for others. The score of major (*Major*) is 1 if the major is software-related, 0 otherwise, and so is the score of Job (*Job*). We divide the duration into several intervals, and the score of duration (*Duration*) is 1 if the developer's work duration is in [0, $dura_1$), 2 if it is in [$dura_1$, $dura_2$), and so on.

$$Edu = \sum_{i=1}^{n} Degree_i \times Major_i \quad (6)$$

$$Wor = \sum_{i=1}^{n} Job_i \times Duration_i \quad (7)$$

2) *"CreGua"*

"*CreGua*" includes credit and guarantee. The original "*CreGua*$_0$" is calculated by (8). Suppose that *CREGUA* is the maximum value among all the enrolled developers, we divide "*CreGua*$_0$" by *CREGUA* to get "*CreGua*".

$$CreGua_0 = Cre \times Cre_w + Gua \times Gua_w \quad (8)$$
$$\text{where } Cre_w + Gua_w = 1$$

"*Cre*" consists of two parts, good record and bad record. The weight of bad record (Bad_w) is greater than weight of good record ($Good_w$), because we punish those developers who have "bad" record, since employers would not deliver tasks to developers with bad credits. As the credit is time-related and reward-related, we introduce two parameters, $Time_w$ and $Reward_w$. $Time_w$ is calculated by (9), where T represents the months of age of the crowdsourcing platform, and Δt represents the period (months) for the evaluation. This formula ensures that the longer from now, the less effect the credit record has. The effect won't disappear. $Reward_w$ is calculated using the similar method as "*Duration*" but the intervals are divided by reward. Since there can be many credit records, "*Cre*" of a developer is calculated by (10).

$$Time_w = (T-\Delta t)/T \times e^{-\Delta t/T} \quad (9)$$

$$Cre = \sum_{i=1}^{n} Good_w \times Time_{wi} \times Reward_{wi} - \sum_{i=1}^{m} Bad_w \times Time_{wi} \times Reward_{wi} \quad (10)$$

"*Gua*" presents the guarantee of a developer. It is measured by the security deposit of a developer. "*Gua*" is calcualted using the similar method as "*Duration*" but the intervals are divided by security deposit.

3) *"Rec"*

The recent performance often reflects the activeness of a developer. The more active a developer is, the more effort can be put in crowdsourcing. "*Rec*" is calculated by combing the number of finished tasks ("*Tas*") of the developer, and earnings he/she has made ("*Ear*") in the recent 3 months. The original "*Rec*$_0$" is calculated by (11), where "*Tas*" and "*Ear*" are calculated using the similar method as "*Duration*" but the intervals are divided by number of recent finished tasks and recent earnings, respectively. Suppose that *REC* is the maximum value of all enrolled developers, we divide "*Rec*$_0$" by *REC* to get "*Rec*".

$$Rec_0 = Tas \times Tas_w + Ear \times Ear_w \quad (11)$$
$$\text{where } Tas_w + Ear_w = 1$$

V. Evaluation Plan

In the future, we will evaluate our approach in the three steps:

- Data preparation. We will fetch the most suitable data of accomplished software development tasks and do preprocessing.
- Parameter tuning. We will use greedy search to find the best value of the parameters in our approach.
- Precision comparison. We will recommend developers to tasks with our approach and compare the precision of our approach with a general approach.

VI. Conclusions

There exists a challenge of assigning the most suitable developers to specific tasks in software crowdsourcing. In this paper, we proposed an approach of recommending developers for software crowdsourcing. Firstly, our approach models the task and the worker comprehensively. Then our approach recommends developers for software crowdsourcing tasks based on their developer-task competence, task-task similarities and developer's soft powers.

Acknowledgment

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242, 91118004).

References

[1] J. Howe, "The rise of crowdsourcing," *Wired*, pp. 1-4, June 2006.

[2] A. Brew, D. Greene, and P. Cunningham, "Using crowdsourcing and active learning to track sentiment in online media," in *Proc. 19th European Conf. on Artificial Intelligence*. Lisbon, 2010, pp. 145–150.

[3] O. F. Zaidan and C. Callison-Burch, "Crowdsourcing translation: Professional quality from non-professionals," in *Proc. 49th Annual Meeting of the Association for Computational Linguistics*. Portland, 2011, pp. 1220-1229.

[4] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H.R. Motahari-Nezhad, E. Bertino, and S. Dustdar, "Quality Control in Crowdsourcing Systems: Issues and Directions," *IEEE Internet Computing*, Vol. 17, pp. 76-81, March-April 2013.

[5] U. Hassan and E. Curry, "A Capability Requirements Approach for Predicting Worker Performance in Crowdsourcing," in *9th Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing*. Austin, 2013, pp. 429-437.

[6] I. Boutsis and V. Kalogeraki, "Crowdsourcing under Real-Time Constraint," in *IEEE 27th Int. Symp. on Parallel & Distributed Processing*. Boston, 2013, pp. 753-764.

[7] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux, "Pick-A-Crowd: tell me what you like, and I'll tell you what to do," in *WWW 2013*. Rio de Janeiro, 2013, pp. 367-374.

[8] E. Simpson and S. Roberts, "Bayesian Methods for Intelligent Task Assignment in Crowdsourcing Systems," in *Studies in Computational Intelligence*, Springer, vol. 538, pp 1-32, Feburary 2015.

[9] B. Satzger, H. Psaier, D. Schall, and S. Dustdar, "Auction-based crowdsourcing supporting skill management," in *Information Systems*, vol.38, pp. 547-560, June 2013.