


Article

# A Data-Driven Game Theoretic Strategy for Developers in Software Crowdsourcing: A Case Study

Zhifang Liao <sup>1</sup>, Zhi Zeng <sup>1</sup>, Yan Zhang <sup>2</sup> and Xiaoping Fan <sup>3,\*</sup>

<sup>1</sup> School of Software, Central South University, Changsha 410075, China; zfliao@csu.edu.cn (Z.L.); zengcs@csu.edu.cn (Z.Z.)

<sup>2</sup> School of Engineering and Built Environment, Glasgow Caledonian University, Glasgow G4 0BA, UK; yan.zhang@gcu.ac.uk

<sup>3</sup> Hunan University of Finance and Economics, Changsha 410075, China

\* Correspondence: xpfan@csu.edu.cn; Tel.: +86-0731-8265-6877

Received: 2 January 2019; Accepted: 15 February 2019; Published: 19 February 2019



**Abstract:** Crowdsourcing has the advantages of being cost-effective and saving time, which is a typical embodiment of collective wisdom and community workers' collaborative development. However, this development paradigm of software crowdsourcing has not been used widely. A very important reason is that requesters have limited knowledge about crowd workers' professional skills and qualities. Another reason is that the crowd workers in the competition cannot get the appropriate reward, which affects their motivation. To solve this problem, this paper proposes a method of maximizing reward based on the crowdsourcing ability of workers, they can choose tasks according to their own abilities to obtain appropriate bonuses. Our method includes two steps: Firstly, it puts forward a method to evaluate the crowd workers' ability, then it analyzes the intensity of competition for tasks at Topcoder.com—an open community crowdsourcing platform—on the basis of the workers' crowdsourcing ability; secondly, it follows dynamic programming ideas and builds game models under complete information in different cases, offering a strategy of reward maximization for workers by solving a mixed-strategy Nash equilibrium. This paper employs crowdsourcing data from Topcoder.com to carry out experiments. The experimental results show that the distribution of workers' crowdsourcing ability is uneven, and to some extent it can show the activity degree of crowdsourcing tasks. Meanwhile, according to the strategy of reward maximization, a crowd worker can get the theoretically maximum reward.

**Keywords:** software crowdsourcing; reward; game; Topcoder.com

## 1. Introduction

Crowdsourcing is used to find high-quality workers in the masses through the Internet; workers can get paid when they complete tasks. Crowdsourcing makes use of people's cognitive advantages to solve difficult problems that computers are unable to, such as data collection and data analyses [1,2] for data mining and knowledge acquisition [3]; it can be used to select high-quality workers from a large number of people at a small cost. Furthermore, such services from people of the group only cost a small amount of money. Software development is rarely done in an isolated environment [4,5]; instead, it increasingly depends on the collaboration among different groups of stakeholders. However, such cooperative development processes frequently face challenges from rapidly evolving requirements, conflicting stakeholders needs and constraints, as well as other factors. In software development, on the one hand, crowdsourcing is attributed with significant cost-savings and quicker task completion time; on the other hand, the crowdsourcing platforms allow software

companies to recruit excellent software developers; therefore, software crowdsourcing has achieved a rapid growth in recent decades. Crowdsourcing platforms emerge one after another, such as Topcoder.com, zbj.com, make8.com, epwk.com, and so on. However, for online workers (they seek software-related part-time work on the web to get paid; most of these tasks are coding, software requirements analysis, code review, and software integration; they communicate through the Internet, not offline), people always lack a "face-to-face" meeting and the confidence that stems from that; at the same time, as the workers' performance is not clear to see, the crowdsourcing platforms and the requesters who host crowdsourcing tasks do not know who is better. As a result, requesters cannot get better software crowdsourcing services. Some previous studies have shown that some popular workers' attributes, such as geographical location and education level, can be used to ascertain the workers' trustworthiness [6–8]. Mok, R.K.P. et al. [9] presented a unique approach to identify low quality workers via QoE crowdtesting and conducted QoE assessment on crowdsourcing Amazon Mechanical Turk and CrowdFlower, through video, so as to collect worker behaviors. Hina Gul Afridi and Mohammad Imran Faisal [10,11] used linear regression to analyze the factors affecting task compensation and workers' reputation, and only the attributes of crowdsourcing tasks were analyzed. These kinds of studies usually only took unilateral factors into consideration, and few of them considered the correlation between crowdsourcing platforms and crowd workers. At the same time, they have not put forward any good solution to solve the most concerning problem, that is, the reward issue. Therefore, studies on crowd workers can help software crowdsourcing platforms fully understand the workers' capacities and their sensitivity to tasks, which will aid managers to design better crowdsourcing strategies. Meanwhile, as for the most concerning issue, about reward, we analyze the intensity of competition for crowdsourcing tasks on the basis of crowd workers' capacities, and provide a strategy of reward maximization for workers, a win-win purpose for both platforms and workers. The paper proceeds from the crowdsourcing capacity and puts the priority on the strategies of reward maximization at different levels of competition intensity.

The rest of the chapters of the paper are organized as follows: The second chapter introduces the related work; the third chapter elaborates on the method design; the fourth chapter shows the experimental data and the analysis of results; and the fifth chapter draws a conclusion.

## 2. Background and Related Work

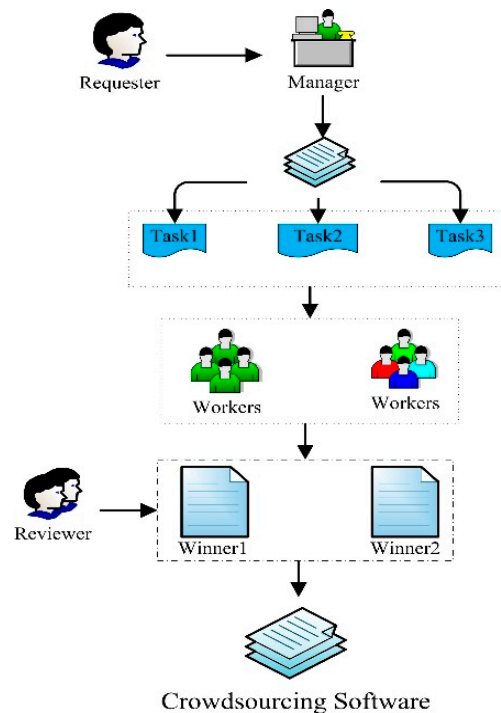
### 2.1. Crowdsourcing in Software Engineering

A large number of netizens are not only users of various software services, but they also make great contributions to various types of network data and service applications, which makes software crowdsourcing possible. For an open network environment, how to effectively manage and coordinate the use of the crowd to maximize the benefits is important, especially in software engineering [12,13].

With the application of crowdsourcing techniques in software engineering [14,15], this has led to the emergence of a new field, named crowdsourcing software engineering (CSE). LaToza and van der Hoek [16] proposed three CSE (i.e., peer production, competitions, and micro-tasking) models based on key factors, such as the number of workers and the expected time needed to solve the issue at hand. Harel D et al. [17] raised a development environment that allows the crowd to observe the work plan of the earlier version and then contribute their wisdom, in a voting-type platform, to achieve the purpose of affecting the program structure and process. Winkler D et al. [18] introduced a crowdsourcing-based inspection (CSI) process with tool support, with the focus on inspection teams and the quality of defect detection. Guzman E et al. [19] used the software and demand evolution information contained in tweets, and presented a survey, with 84 software engineering practitioners and researchers, that studies the tweet attributes that are most telling of tweet priority when performing software evolution tasks. However, at present, the crowd research on the crowdsourcing platform is limited. Our work focuses on the research of workers on Topcoder.com [20].

## 2.2. Topcoder.com

Topcoder.com is a famous software crowdsourcing platform with more than one million registered users, seven thousand challenges hosted per year, and up to 80 million dollars in challenge payouts and high prizes, and a large number of tasks provide data support for our research. It also has its own community and social networks; members compete with each other and cooperate to gain money and honors [21]. In this paper, we use Topcoder.com as a case study to study the workers' abilities. Figure 1 is the flow chart of the crowdsourcing software development (CSD) mechanism.



**Figure 1.** The flow chart of the crowdsourcing software development mechanism. Manager decomposes a large software project to several components as requested, posts these components on Topcoder.com, and get high quality solutions by crowdsourcing.

As shown in Figure 1, according to the requirements, the platform decomposes a large software project to several components, posts these components on Topcoder.com as tasks, and obtains high quality solutions through competition. For online workers, they choose their tasks of interest to get payment. Workers need to submit their own solutions online, which will be reviewed next if some of them meet the requirements. During the review, three reviewers will evaluate the workers' submissions and give scores on "scorecards". The examination covers contents such as whether the submission meets the requirements, whether the interface design is beautiful and friendly, whether the code is concise, and whether the corresponding code comments are given. After reviewers publish scores, a worker can revise his or her submission to get a higher score. By doing this repeatedly, the final score will be obtained. According to the final scores, generally the top three will be paid a large amount of money and will gain the corresponding Topcoder.com points. With the accumulation of points, developers with top-ranking points will be more likely to get future crowdsourcing opportunities and considerable rewards. The whole software development process, architecture, prototyping, coding, testing, and even code reviews are all assigned to the platform as crowdsourcing tasks. Crowd workers choose tasks on the basis of their own abilities and professional skills.

### 2.3. Research on Crowd Workers

Some of the existing studies on developers in crowdsourced software development focus on the statistical analysis of developers' behavior data [22–24]. In the literature [25], Dubey A et al. analyzed crowd developer's characteristics on Topcoder.com and upwork.com to predict the quality of tasks completed by them. Furthermore, they put forward the trustworthiness of software crowdsourcing to help enterprises get rid of certain risks [26]. Alelyani T and Yang Y [27] discussed the reliability that different workers showed on different tasks, so as to find out certain behavioral characteristics of the crowd developers. Hu Z and Wu W [28,29] studied the offense–defense mechanism of Topcoder.com contests, constructed a game theoretical model, and calculated the Nash equilibrium to deduce these developers' specific behavior decisions. Kuang L, Zhu Y, Li S, et al. [30] used game theory in data privacy protection. Other studies paid attention to the correlation between developers and the reward of tasks. Hina Gul Afridi [11] found, by employing multiple linear regression, that on the contest-based software crowdsourcing platform, a developer's incentive to participate was highly correlated with the reward of the task. Mohammad Imran Faisal [10] also used linear regression to find out the factors that affect tasks and predict the quality of the crowdsourcing tasks. Research by Machado L et al. [31] showed that Topcoder.com's contest mechanism is a great challenge for developers who were used to traditional software development, when they engage themselves in software crowdsourcing for the first time, indicating that crowdsourcing development is not friendly to the newcomer. For them, contest-based incentive mechanism puts them on the unfavorable side, whereby they cannot get suitable reward, leading to a loss of fresh blood for the platforms.

In conclusion, in the studies on crowd workers, workers tend to be concerned about their reward the most. Therefore, analyzing and studying the worker's reward strategy can not only help the platform to distribute crowdsourcing tasks in a reasonable and proper way, but also help the workers choose their suitable tasks.

## 3. Research Design

### 3.1. Study Overview

In fact, when there are many workers involved in the same task and the “top three win prizes”, this is actually based on the competitive incentive model; but there is also a problem here, because if the workers' ability and numbers are different, the intensity of competition is also different. Therefore, for the worker, choosing a task becomes important because he/she wants to get paid, so he/she needs to consider the ability of other participants. In order to maximize a worker's reward, the ability of workers and the competition are the key factors.

In order to study the strategy of maximizing the worker's reward, a model of crowdsourcing ability assessment is proposed in this paper. This model firstly analyzes the competition intensity of tasks on the Topcoder.com platform. According to different task intensities, we use the knapsack algorithm and solve the mixed-strategy Nash equilibrium to obtain the strategy of maximized reward, providing references for workers when they choose tasks. There are three major contributions in this paper.

#### 3.1.1. Build a Model of Crowdsourcing Ability Assessment for Crowd Workers

Combined with many factors affecting the workers' success, we build a model of crowdsourcing ability assessment, with the aim of getting a more visual understanding of the workers' comprehensive performance in completing crowdsourcing tasks of various types.

#### 3.1.2. Analyze the Competition Intensity of Tasks Posted on a Crowdsourcing Platform

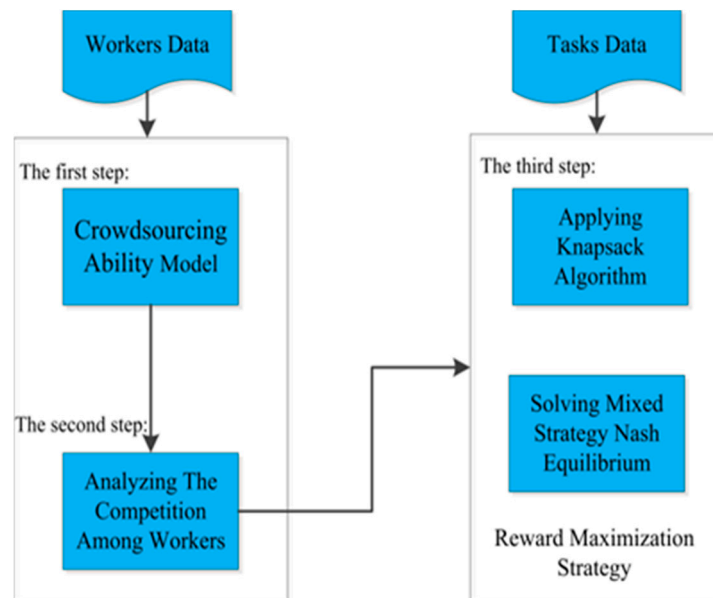
The main purpose of this work is to get workers better acquainted with their sensitivity to crowdsourcing tasks, and help the platform to work out a strategy of proper task distribution at the same time.

### 3.1.3. Research on the Strategy of Reward Maximization for Workers

According to the competition intensity of tasks, this paper carries out modeling analysis of the reward maximization. In consideration of each worker’s actual situation, it offers the strategies of reward maximization to targeted developers. The research steps are shown in Figure 2. In the research on workers’ crowdsourcing ability, most of the symbols and attributes used in formula definition, derivation, and algorithm construction are shown in Table 1.

**Table 1.** Notations used in crowdsourcing ability model.

Symbol	Description
$T_p(w_i, v_j)$	The worker $i$ 's participation in tasks type $j$ .
$SR(w_i, v_j)$	The submission ratio of a worker $i$ in task type $j$ . It is obtained by dividing the number of submissions divided by the number of participants.
$S(w_i, v_j)$	Number of worker $i$ 's submissions in task type $j$ .
$S_q(w_i, v_j)$	The overall quality of the worker $i$ 's submission in task type $j$ .
$PR(w_i, v_j)$	Number of worker $i$ 's pass reviews in task type $j$ .
$RR(w_i, v_j)$	The pass review ratio of a worker $i$ in task type $j$ . It is obtained by dividing the number of reviews by the number of submissions.
$O_r(w_i, v_j)$	The worker $i$ 's overall ranking in task type $j$ .
$WP(w_i, v_j)$	The win ratio of worker $i$ in task type $j$ . It is obtained by dividing the number of wins by the number of reviews.
$W(w_i, v_j)$	Number of worker $i$ 's wins in task type $j$ .
$Ap(w_i, v_j)$	The average placement of worker $i$ in task type $j$ .
$CA(w_i, v_j)$	The crowdsourcing ability of worker $i$ in task type $j$



**Figure 2.** Research steps: Step one, building the crowdsourcing ability model; step two, analyzing the competition among workers; and step three, obtaining reward maximization strategy by using knapsack algorithm and constructing game models.

### 3.2. Methodology

This article mainly carries out research in the following three aspects.

### 3.2.1. Build a Model of Crowdsourcing Ability Assessment for Crowd Workers

For the crowd workers, the process of crowdsourcing software development is to find the task distributed by the platform, register the task, submit programs, review of programs, publish the notice, and collect the reward. In the process, the worker's work of each stage can directly determine whether he or she can win the monetary reward in the end. In order to build a model of crowdsourcing ability assessment for workers, we collected the worker's information at different stages. According to the development process, a worker's crowdsourcing ability mainly shows in three aspects: 1) the ratio of true participation in the crowdsourcing task, 2) the overall quality of the submissions, and 3) the worker's overall ranking. Before building the model, we needed to define some indicators.

**Definition 1.** *Ratio of true participation, an indicator to measure the true involvement of a worker in crowdsourcing development, rather than merely registering tasks and being an observer. The ratio of true participation is calculated as below:*

$$T_p(w_i, v_j) = SR(w_i, v_j) \times S(w_i, v_j) \quad (1)$$

In Equation (1),  $SR(w_i, v_j)$  stands for the ratio of solutions submitted to crowdsourcing tasks of different types (generally the crowdsourcing tasks fall into four types: Code, F2F, Assembly, Architecture) of the worker  $i$ , where the ratio of true participation is in direct proportion to submission ratio and times of participation, which can not only measure the worker's attention paid to the crowdsourcing tasks, but also ensure that his participation times are sufficient.

**Definition 2.** *The overall quality of the submissions, the main purpose of which is to judge whether the crowdsourcing tasks are completed in good quality. The specific calculation is shown in Equation (2).*

$$S_q(w_i, v_j) = PR(w_i, v_j) \times RR(w_i, v_j) \quad (2)$$

In Equation (2),  $RR(w_i, v_j)$  stands for the ratio of the worker  $i$ 's submissions that have passed the review when he/she takes part in tasks of certain types. We can see that the overall quality is the product of the number of submissions that have passed the review, multiplying the ratio of submissions passed the review. The use of product was to avoid the consequence that a single attribute may have a great effect on the results.

**Definition 3.** *The overall ranking. It can, most directly, show the worker's winning situation in a certain type of crowdsourcing task, and indirectly reflect the worker's overall income. The calculation is shown as below:*

$$O_r(w_i, v_j) = WP(w_i, v_j) \times W(w_i, v_j) \quad (3)$$

In Equation (3),  $WP(w_i, v_j)$  represents the proportion of the crowd worker's submissions, from the passage of the review to winning the final.  $W(w_i, v_j)$  represents the number of times that the worker has won the crowdsourcing tasks. After some indicators were well defined, the model of crowdsourcing ability assessment was calculated in detail, as below:

$$CA(w_i, v_j) = \frac{\sqrt{Tp(w_i, v_j)^2 + Sq(w_i, v_j)^2 + Or(w_i, v_j)^2}}{Ap(w_i, v_j) + \mu(w_i, v_j)} \quad (4)$$

In Equation (4),  $Ap(w_i, v_j)$  is the worker's average ranking. In the process of building the model, we found out that the average ranking of some workers had been lost. In order to eliminate the assessment inaccuracy caused by the loss of average rankings, a parameter  $\mu(w_i, v_j)$  was employed. If the crowd worker had the average ranking, it was zero; if not, its value would be replaced by

the average value of the ranking he/she earned in participating crowdsourcing tasks, which was calculated as below:

$$\mu(w_i, v_j) = \frac{\sum_d^{D(v_j)} Ns(w_i, v_j)}{\sum_d^{D(v_j)} n(w_i, v_j)} \tag{5}$$

In Equation (5),  $Ns(w_i, v_j)$  stands for the number of the crowd worker's submissions,  $n(w_i, v_j)$  is the times he/she takes part in completing crowdsourcing tasks, and  $D(v_j)$  means type  $j$  tasks in all crowdsourcing tasks.

### 3.2.2. Research on the Competition Relations among the Crowd Workers

After the model was built, analysis of the competition among the workers on the basis of workers' ability can help online developers choose suitable crowdsourcing tasks. Firstly, we classified workers into different groups via the crowdsourcing ability model and analyzed the competition intensity in detail. When classifying workers, we considered the points ranking information on Topcoder.com. The competition index for worker  $n$  and worker  $m$  in completing crowdsourcing tasks of type  $v_j$  was calculated as follows:

$$CI(w_n, w_m) = \frac{CA(w_n, v_j) - CA(w_m, v_j)}{CA(w_n, v_j) + CA(w_m, v_j)} \tag{6}$$

According to the calculation, when the absolute value of competition index was closer to zero, the competition was fiercer; when it was closer to one, the competition was weaker; when there is only one worker choosing the crowdsourcing task, there is no competition. The analysis of competition relations is the precondition of the strategy of reward maximization. When the competition was weak, the worker can only took his own actual situation into consideration in decision-making, which means he/she has more freedom to choose crowdsourcing tasks. However, when the competition was fierce, the worker not only needed to consider his own situation, but also to consider if the participation of other competitors affects the final results and the prize distribution.

### 3.2.3. Research on the Strategy of Reward Maximization for Workers

After the analysis of the workers' ability and their competition relationship, we built a model of the strategy of reward maximization. When the workers win, they will get money and points as remuneration. According to the competition analysis, the optimal strategy is closely correlated to the competition intensity. We analyzed two separate situations according to the competition intensity and the detailed information is described as below.

#### A. The Competition is Weak.

Before discussing the reward strategy, we carry out reasonable assumptions and describe the question.

**Assumption 1.** *When the competition was weak, developers with a strong crowdsourcing ability can get rewarded. At the same time, crowdsourcing tasks were relatively abundant during this period (in fact, crowdsourcing tasks are added dynamically).*

**Question 1.** *When the number of crowdsourcing tasks is  $n$ , completion of crowdsourcing task  $i$  consumes the time  $t_i$ , and its reward is  $v_i$ , and the duration of the contest season is  $T$ , then how would one choose the best crowdsourcing task during a certain period if the worker wants to get the most reward?*

*Formal description:* Given  $T > 0, t_i > 0, v_i > 0, 1 \leq i \leq n$ , we need to find  $n$  vectors, in which  $x_i \in \{0, 1\}$ , satisfying the condition  $\sum_{i=1}^n t_i x_i < T$  to make the  $\sum_{i=1}^n v_i x_i$  the largest, namely:

$$\max \sum_{i=1}^n v_i x_i \tag{7}$$

$$s.t \begin{cases} \sum_{i=1}^n t_i x_i < T \\ x_i \in \{0, 1\}, 1 \leq i \leq n \end{cases} \quad (8)$$

From the question description we can see that in weak competition the strategy of reward maximization for workers is the 0–1 knapsack problem. Dynamic programming can be used to figure out the equation, and the process of which includes two statuses:

Status 1: There remains time after the  $i - 1$  tasks have been chosen, in which case the maximized income can be acquired without choosing the  $i$ th task.

Status 2: There remains time  $T - t_i$  after choosing the task  $i - 1$ , in which case choosing the  $i$ th task can help him get the maximum reward. The state transition equation for this problem is:

$$f(i, T) = \max\{f(i - 1, T), f(i - 1, T - t_i) + v_i\} \quad (9)$$

When the competition was weak, the pseudocode of the maximized reward algorithm was written below. Generally, the time  $T_i = \{t_1, t_2, \dots, t_i\}$  consumed by completing tasks, reward  $V = \{v_1, v_2, \dots, v_i\}$ , and the whole time  $T$  consumed by the worker were taken as the input. The algorithm will output the vector  $x = \{x_1, x_2, \dots, x_i\}$ , the task list chosen by the worker when he/she earns most. The reward maximization algorithm falls into two parts: The first part figures out the worker's maximized reward and the second part computes the tasks that should be chosen.

---

#### The Algorithm of Reward Maximization

---

Input: The cost of tasks,  $T_i = \{t_1, t_2, \dots, t_i\}$ ; the prize of tasks,  $V = \{v_1, v_2, \dots, v_i\}$ ; total time:  $T$

```

1.  $n \leftarrow \text{length}[V]$ 
2. for  $j \leftarrow 0$  to  $T$ 
3.   do  $m[0, j] \leftarrow 0$ 
4. for  $i \leftarrow 1$  to  $n$ 
5.   do  $m[i, 0] \leftarrow 0$ 
6.     for  $j \leftarrow 1$  to  $T$ 
7.       do  $m[i, j] \leftarrow m[i - 1, j]$ 
8.         if  $t_i \leq j$ 
9.           then if  $v_i + m[i - 1, j - t_i] > m[i - 1, j]$ 
10.            then  $m[i, j] \leftarrow v_i + m[i - 1, j - t_i]$ 
11. return  $m$ .
12.  $n \leftarrow \text{length}[T_i]$ 
13.    $j \leftarrow T$ 
15. for  $i \leftarrow n$  to 1
16. do if  $m[i, j] = m[i - 1, j]$ 
17.   then  $x[i] \leftarrow 0$ 
19. else  $x[i] \leftarrow 1$ 
20.    $j \leftarrow j - t_i$ 
21. return  $x$ 

```

Output:  $x = \{x_1, x_2, \dots, x_i\}$ ,  $x \in \{0, 1\}$ , made  $\sum_{i=1}^n t_i x_i < T$  and  $\sum_{i=1}^n v_i x_i$  is the maximum.

---

In this algorithm, the time complexity of line two to line three is  $O(n)$ . Two nested for-loops within line four to line 10 repeats  $n$  times in the outer layer and  $T$  times in the inner layer, of which the time complexity is  $O(Tn)$ . The time complexity of the rest of the lines is  $O(n)$ , obviously.

#### B. The Competition is Strong.

When the competition was strong, the crowd workers' individual interests in maximization turns to be a game problem. The optimal strategy can be obtained by constructing a two-player game model, within the context of complete information and solving the mixed-strategy Nash equilibrium.



Same as for the above, before we discuss the reward strategy, firstly we carry out the reasonable assumptions and describe the question.

**Assumption 2.** When the competition was strong, there was a certain probability that worker A and worker B, both with a strong crowdsourcing ability, can get rewarded. At the same time, they knew each other very well (in fact, the workers can communicate with each other on the Topcoder.com community), and the crowdsourcing tasks were relatively abundant in this period (in fact, crowdsourcing tasks are added dynamically).

**Question 2.** When worker A and worker B are faced with the same crowdsourcing task, their winning probabilities are  $P_a$  and  $P_b$ , respectively, the times needed to be consumed are  $t_a$  and  $t_b$  respectively, and the reward they get is R. In fact, the reward is divided into two forms: payment (the top is P and the second is p) and score (S). The distribution of the score is adjusted by the parameter (in fact Topcoder.com sets it as 0.7, that is, the top worker gets a 70% score and the second gets the rest). What are the probabilities of worker A and worker B getting the maximum reward if they take part in the same task?

Formal description: In a game  $G = (S_1, \dots, S_n; u_1, \dots, u_n)$  of a crowdsourcing task participated by  $n$  developers, the mixed-strategy profile constructs a  $p^* = \{p_1^*, \dots, p_i^*, \dots, p_n^*\}$  Nash Equilibrium, for all  $i = 1, 2, \dots, n$  the formula below holds:

$$v_i(p_i^*, p_{-i}^*) \geq v_i(p_i, p_{-i}^*), \forall p_i \in \sum_i p^* \tag{10}$$

The strategy forms a Nash equilibrium if one strategy group makes the strategy of any participant the best compared to other participants' strategies. Here we focus on the analysis of the two-player game when workers choose the task at Topcoder.com. The pay-off matrix constructed with complete information is shown in Table 2.

Table 2. Pay-off matrix.

	$B_1(\theta)$	$B_0(1 - \theta)$
$A_1(\mu)$	$(\pi(A_{11}), \pi(B_{11}))$	$(\pi(A_{10}), 0)$
$A_0(1 - \mu)$	$(0, \pi(B_{10}))$	$(0, 0)$

In Table 2,  $\pi(A_{11})$  stands for A's reward and  $\pi(B_{11})$  for B's reward when both worker A and B participate;  $\pi(A_{10})$  stands for A's reward when A is in the contest while B is not, and  $\pi(B_{10})$  represents B's reward when B is in while A is not. Each reward is calculated as follows:

$$\pi(A_{11}) = P_a(P + \alpha S) + (1 - P_a)(p + (1 - \alpha)S) - t_a \tag{11}$$

$$\pi(B_{11}) = P_b(P + \alpha S) + (1 - P_b)(p + (1 - \alpha)S) - t_b \tag{12}$$

$$\pi(A_{10}) = P + S - t_a \tag{13}$$

$$\pi(B_{10}) = P + S - t_b \tag{14}$$

After the calculation of each reward, the mixed Nash Equilibrium in this game is figured out. The reward functions of the worker A and worker B are shown as below, respectively:

$$E_A(\mu, \theta) = \mu[\theta(\pi(A_{11})) + (1 - \theta)(\pi(A_{10}))] \tag{15}$$

$$E_B(\mu, \theta) = \theta[\mu(\pi(B_{11})) + (1 - \mu)(\pi(B_{10}))] \tag{16}$$

As reward is calculated in this way: A's reward when both A and B participate plus A's reward when A participate while B does not. After these calculations are done, the next step is to work out  $\mu$

and  $\theta$ . Here we use the method of payment maximization to solve the Nash Equilibrium of the game. Steps are as follows:

In the reward functions of A, firstly we figure out the partial derivative of  $\mu$ :

Let  $f_A(x) = \frac{\partial E_A(\mu, \theta)}{\partial \mu}$ , we can get the maximum value of  $E_A(\mu, \theta)$  provided  $f_A(x) = 0$ , then  $\theta$  can be worked out as below:

$$\theta = \frac{P + S - t_a}{(P - p)(1 - P_a) + S(\alpha - (2\alpha - 1)P_a)} \tag{17}$$

Similarly, in reward functions of B, we figure out the partial derivative of  $\theta$ ,

Let  $f_B(x) = \frac{\partial E_B(\mu, \theta)}{\partial \mu} = 0$ , and we arrive at:

$$\mu = \frac{P + S - t_b}{(P - p)(1 - P_b) + S(\alpha - (2\alpha - 1)P_b)} \tag{18}$$

After the value of  $\mu$  and  $\theta$  are figured out, let  $\lambda = (\mu, 1 - \mu)$  and  $\varphi = (\theta, 1 - \theta)$  be the mixed Nash Equilibrium strategies for worker A and B, respectively, and  $(\mu, \theta)$  is the Nash Equilibrium of the game. Besides, in the process of solving the game, the method of payment equivalents can also be used to solve the game.

### 4. Empirical Results

#### 4.1. Dataset

Two datasets from Topcoder.com were mainly used in this paper: One was the personal information of the crowd workers and the other was the information of the crowdsourcing tasks. As a well-known software crowdsourcing website, Topcoder.com has a large amount of crowdsourcing data. In regard to crowd workers' personal information, we crawled the information of 2176 developers who were active in the past two years. In regards to the collection of crowdsourcing task data, we crawled and organized all the crowdsourcing tasks on Topcoder.com, including all types of crowdsourcing tasks and results of some crowdsourcing tasks. The details of datasets are shown in Tables 3 and 4.

**Table 3.** The dataset of pass challenges' attributes and descriptions.

Attributes	Description
Challenge Type	The type of tasks, including specification, architecture, and assembly
Num Registrants	Number of registrants. Range (0, + ∞)
Num Submissions	Number of submissions where registrants submit their solutions. Range (0,+ ∞).
First Place Prize	Monitory prize will be given to the top (one) solution.
Registration Start Date	Time that tasks were posted in Topcoder.com.
Registration End Date	Deadline that all workers who registered for a task must submit their final results.

**Table 4.** The dataset of handler details' attributes and descriptions.

Attributes	Description
Competitions	Number of this worker's registrations.
Submissions	Number of this worker's submissions.
Pass review	Number of this worker's passed reviews.
Pass screening	Number of this worker's passed screenings.
Wins	Number of this worker's task wins.
Average placement	The average placement of this worker.

In data processing, firstly we needed to remove the crowdsourcing tasks that had been canceled or had not been completed due to various reasons; secondly, we needed to remove the crowd workers who had never registered or had never submitted a program. Then we analyzed the raw data and removed the attributes that had no effect on the study from the dataset. In dealing with Leaderboard

points, we summed up the points gained by the same crowd worker in completing different types of crowdsourcing tasks, so as to do intermediate experiments later.

#### 4.2. Experiments

In order to analyze the workers' crowdsourcing ability and the competition intensity at Topcoder.com, and find out the factors influencing the optimal strategy for developers, we mainly conducted three experiments, as follows.

##### 4.2.1. Experiment on the Model of Crowdsourcing Ability Assessment for Crowd Workers

In order to study the workers' crowdsourcing abilities, we evaluated the validity of crowdsourcing capacities based on the crowdsourcing assessment model. In this section, we conducted the following two experiments: The first was to find out the relationship between the number of developers and the crowdsourcing capacity; the second was to compare the rankings of calculated crowdsourcing ability with the point rankings of developers at the Topcoder.com website.

In the experiment on the model of crowdsourcing ability assessment, according to the crowdsourcing tasks preferred by the 2176 workers, we worked out their capacities to complete crowdsourcing tasks in the category through the crowdsourcing assessment model we built. Figure 3 shows the relationship between crowdsourcing capacity and the number of the workers.

First of all, we used the dataset called "handler details". From Figure 3 we can see that the relationship between crowdsourcing capacity and the number of developers was quite consistent with the long-tailed distribution, indicating that most of developers were observers, meaning they just register tasks at will, simply submit the solutions, and ultimately, for various reasons, they do not engage themselves in long-term software development.

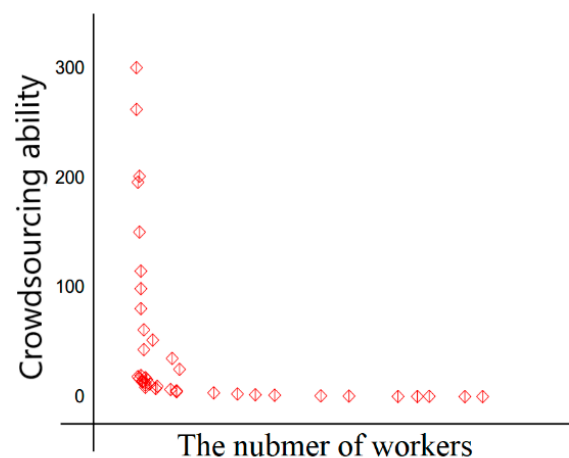
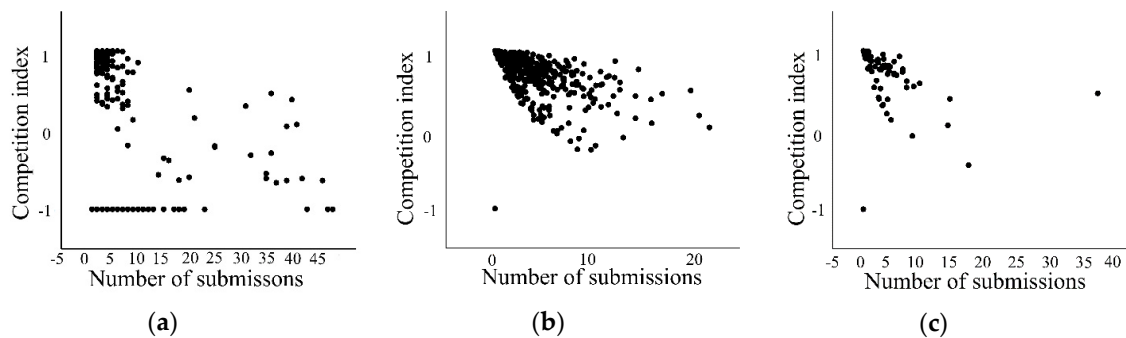


Figure 3. The distribution of workers' crowdsourcing capacities.

However, as time goes by, and due to workers' personal reasons, some long-term contributors may leave. To illustrate the issue of the outflow of developers, we conducted an intermediate experiment. We ranked the developers after the calculation of their crowdsourcing capacities, and then used the point ranking data collected from the Topcoder.com website to do the matching experiments. During the matching, we found some interesting phenomena. For example, if a worker had a high rank in the crowdsourcing ability list but a very low rank, or no place, in the point ranking list, such developer was often a core contributor who had engaged himself a lot in Topcoder.com's crowdsourcing tasks but was no longer involved for various reasons. On the contrary, if a worker had a high rank in both lists, the worker was often the long-term contributor to software crowdsourcing.

#### 4.2.2. Experiments on the Competition Relationship among the Crowd Workers

According to the intermediate experiment, we believe that the workers that appeared on the point list had a stronger ability, and their CA value was not less than five. Seeing this as a prerequisite, in order to analyze the competition intensity of the crowdsourcing tasks, we divided the crowdsourcing ability into two groups: strong and weak (if the value of crowdsourcing ability is higher than five, the developer belongs to the strong group, and vice versa). The experiments in this section employ the dataset covering crowdsourcing tasks of code type in the past three years. Figure 4 shows the competition index and the number of submissions on Topcoder.com in the past three years.



**Figure 4.** The 2015 (a), 2016 (b), and 2017 (c) competition indices.

From the three figures, we can see that in 2016 and 2017 the competition was fierce and most tasks were competed for. In 2015; however, many tasks were not involved in competitions, that is to say, only a small number of workers were competent or qualified to fulfill the tasks. It indicates that the Topcoder.com managers make the competition fiercer by steering the strategy. Three figures also state that the competition index was not in direct proportion to the number of participants. When the number of participants was around two to eight, the competition was fiercer.

#### 4.2.3. Experiments on the Strategy of Reward Maximization for Workers and Its Analysis

In the experiments of competition relationship, we can see that the competition for crowdsourcing tasks is becoming more and more acute. The situation whereby one worker completes the task and gets the reward alone is becoming less and less. Due to the personal factors of the worker (for example, whether he/she really knows the ability of other workers when selecting tasks) and the number of tasks posted by Topcoder.com.com in different time periods, a case study on Topcoder.com-controlled experiments is affected by many factors. This section mainly analyzes the factors affecting workers when the competition is relatively strong. In the mixed-strategy Nash equilibrium model, the ultimate involvement of a worker in competitions is closely related to the reward, points, winning probability, and other personal reasons. In literature [9], a regression method was used to find out that the worker's incentive to participate was highly related to the reward. As for the workers, the cost is the time consumed by developing. Firstly, we selected the top 20 workers, based on who were good at coding from the points list, and calculated the prize they received per unit time. Results are shown in Figure 5.

As it can be seen from Figure 5, when the average bonus exceeds \$1000, the prize received per unit time by the worker was relatively stable (around \$10~\$18 per hour), implying that their ability to get a prize was relatively stable when they were faced with such complicated tasks. When the average prize was around \$500 to \$1000, the prize received per unit time by the workers displayed violent fluctuations, indicating that the ability to earn a prize turns to unstable, which may be related to the complexity of the tasks and the prize distribution.

Similarly, we can see that when two excellent workers were faced with the same crowdsourcing task of considerable reward, and their ability was evenly matched, then whether to choose the task depends on the extra points given after its completion. For different requests, there are the corresponding crowdsourcing strategies provided. If the worker craves for the offline crowdsourcing

contests and such contests require plenty of points, then the developer may fall over himself to participate in the crowdsourcing tasks with extra points

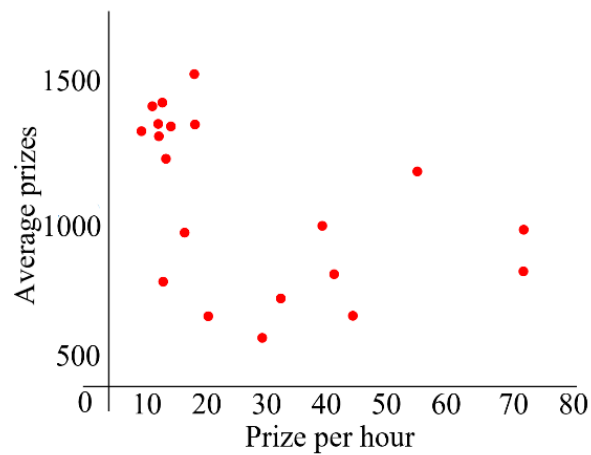


Figure 5. Top 20 average prizes and development times.

When the crowdsourcing tasks offers great money reward, it also means more time will be spent in developing, that is, time elasticity is not enough. Therefore, if the developer only cares about monetary reward and does not want to spend much time, it is recommended that he/she chooses the appropriate, less-paid tasks, without high complexity. By doing this way, he/she can get a large amount of money in a short period and enjoy sufficient time elasticity at the same time. Furthermore, the number of crowdsourcing tasks of such type is much more than the number of tasks with great remuneration. The worker can exert his/her competency completely when those outstanding developers are involved in those tasks with high complexity and higher reward. Figure 6 shows the relationship between the time consumed by submitting solutions and the prize of the tasks.

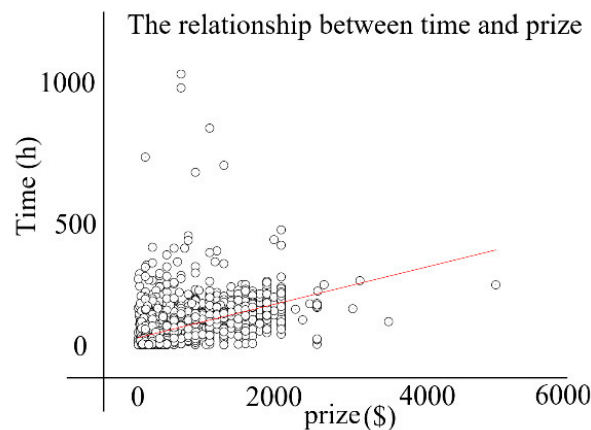


Figure 6. The relationship between time and prize.

As it can be seen from the experimental results in Figure 6, the worker’s development time is positively related to the amount of compensation, to some extent. A contest can stimulate the competition among workers. However, for workers who spent time and yet won no place in ranking, they cannot get any reward, which means the contest model is not friendly to the newcomers.

**5. Conclusions**

In this paper, we first built a model of a crowdsourcing ability assessment for workers and analyzed the relationship of competitions on the basis of crowdsourcing ability. Then, using knapsack algorithm and a game theoretical model, we proposed the optimal strategies for workers when

competition intensity changes. In the end, we used the real data to carry out verification and the main experimental results are written as below:

(1) The relationship between the workers' crowdsourcing ability and the number of crowd workers is in line with the long-tailed distribution. In other words, a relatively small group of developers contribute more solutions to crowdsourcing tasks.

(2) On Topcoder.com, the competition between the workers is getting increasingly fierce, which is a result of the strategy adjusted by the platform managers. However, the acute competition is not friendly to the newcomers, which should be reflected on by the platform managers.

(3) Then, in the research on the strategy of reward maximization for developers, the experiments found that the task reward is the key factor influencing the developer's participation. At the same time, as for the more complicated tasks, the development efficiency of the workers is relatively stable. The developer considers his skills, motivation to participate, and the participation of rivals to choose suitable tasks and get considerable remuneration.

In fact, if the crowdsourcing process of other software crowdsourcing platforms is similar to Topcoder.com, then the worker capability model can be used, because, in this process, workers must participate in the task and submit their own solutions to get paid. If there is competition between workers, only a slight modification of the income function of the game model is needed, and the optimal solution can still be obtained. But for other software crowdsourcing processes, and different incentives, that may not be appropriate. The model of crowdsourcing ability is based on the idea that "excellent workers often have excellent historical information". Through the historical information of workers in the crowdsourcing process, excellent workers could be found. However, the idea based on historical information cannot evaluate new excellent workers, this is a defect of this method. In the future work, we will collect data from different crowdsourcing platforms to study the impact produced by specific incentive mechanisms on the workers, especially the changes of workers' enthusiasm and variations in crowdsourcing capacity within different incentive mechanisms. Combining the methods of machine learning, visualization technology [32], and biological computer [33], we will detect the phenomena of malicious crowdsourcing on several open crowdsourcing communities, based on which, and considering various factors, we will provide customized recommendation across platforms for workers.

**Author Contributions:** All the authors discussed the algorithm required to complete the manuscript. Z.L. and X.F. conceived the paper. Z.Z. performed the experiments and wrote the paper. Y.Z. checked for typos.

**Funding:** The works that are described in this paper are supported by NSF 61876190, Hunan Provincial Key Laboratory of Finance and Economics Big Data Science and Technology (Hunan University of Finance and Economics) and HNSF 2018JJ2535.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Yao, D.; Sun, H.; Liu, X. Combining Crowd Contributions with Machine Learning to Detect Malicious Mobile Apps. In Proceedings of the 7th Asia-Pacific Symposium on Internetware, Wuhan, China, 6 November 2015; pp. 120–123.
2. Sun, H.; Zhang, W.; Yan, M.; Liu, X. Crowdsourcing: Cloud-Based Software Development. In *Recommending Web Services Using Crowdsourced Testing Data*; Springer: Berlin/Heidelberg, Germany, 2015; ISBN 978-3-662-47010-7.
3. Kuang, L.; Yu, L.; Huang, L.; Wang, Y.; Ma, P.; Li, C. A Personalized QoS Prediction Approach for CPS Service Recommendation Based on Reputation and Location-Aware Collaborative Filtering. *Sensors* **2018**, *18*, 1556. [[CrossRef](#)] [[PubMed](#)]
4. Saremi, R.L.; Yang, Y.; Ruhe, G.; Messinger, D. Leveraging crowdsourcing for team elasticity: An empirical evaluation at Topcoder. In Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP '17), Buenos Aires, Argentina, 20–28 May 2017.

5. Liao, Z.; Yi, M.; Wang, Y.; Liu, S.; Liu, H.; Zhang, Y.; Zhou, Y. Healthy or Not: A Way to Predict Ecosystem Healthin GitHub. *Symmetry* **2019**, *11*, 144. [[CrossRef](#)]
6. Kazai, G.; Kamps, J.; Milic-Frayling, N. The face of quality in crowdsourcing relevance labels: Demographics, personality and labeling accuracy. In Proceedings of the 21st ACM International Conference on Information and Knowledge Management, Maui, HI, USA, 29 October–2 November 2012.
7. Kazai, G.; Kamps, J.; Milic-Frayling, N. An analysis of human factors and label accuracy in crowdsourcing relevance judgments. *Inf. Retr.* **2013**, *16*, 138–178. [[CrossRef](#)]
8. Di Palantino, D.; Karagiannis, T.; Vojnovic, M. *Individual and Collective User Behavior in Crowdsourcing Services*; Technical Report; Microsoft Research: Redmond, WA, USA, 2011.
9. Mok, R.K.P.; Chang, R.K.C.; Li, W. Detecting low-quality workers in QoE crowdtesting: A worker behavior based approach. *IEEE Trans. Multimed.* **2017**, *19*, 530–543. [[CrossRef](#)]
10. Faisal, M.I. Predicting the quality of contests on crowdsourcing-based software development platforms: Student research abstract. In Proceedings of the Symposium on Applied Computing (SAC '17), Marrakech, Morocco, 3–7 April 2017; ACM: New York, NY, USA; pp. 1305–1306.
11. Afridi, H.G. Empirical investigation of correlation between rewards and crowdsource-based software developers. In Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C '17), Buenos Aires, Argentina, 20–28 May 2017; IEEE Press: Piscataway, NJ, USA; pp. 80–81.
12. Liao, Z.; Zhao, B.; Liu, S.; Jin, H.; He, D.; Yang, L.; Zhang, Y.; Wu, J. A Prediction Model of the Project Life-Span in Open Source Software Ecosystem. *Mob. Netw. Appl.* **2018**, 1–10. [[CrossRef](#)]
13. Liao, Z.; Deng, L.; Fan, X.; Zhang, Y.; Liu, H.; Qi, X.; Zhou, Y. Empirical Research on the Evaluation Model and Method of Sustainability of the Open Source Ecosystem. *Symmetry* **2018**, *10*, 747. [[CrossRef](#)]
14. Mao, K.; Capra, L.; Harman, M.; Jia, Y. A survey of the use of crowdsourcing in software engineering. *J. Syst. Softw.* **2017**, *126*, 57–84. [[CrossRef](#)]
15. Mao, K.; Yang, Y.; Wang, Q.; Jia, Y.; Harman, M. Developer recommendation for crowdsourced software development tasks. In Proceedings of the 2015 IEEE Symposium on Service-Oriented System Engineering (SOSE), San Francisco Bay, CA, USA, 30 March–3 April 2015; pp. 347–356.
16. LaToza, T.D.; van der Hoek, A. Crowdsourcing in Software Engineering: Models, Motivations, and Challenges. *IEEE Softw.* **2016**, *33*, 74–80. [[CrossRef](#)]
17. Harel, D.; Heimlich, I.; Marelly, R.; Marron, A. Crowd-based programming for reactive systems. In Proceedings of the 2017 International Workshop on Crowdsourcing in Software Engineering, Buenos Aires, Argentina, 22–22 May 2017; IEEE Press: Piscataway, NJ, USA, 2017; pp. 9–13.
18. Winkler, D.; Sabou, M.; Petrovic, S.; Carneiro, G.; Kalinowski, M.; Biffl, S. Improving Model Inspection with Crowdsourcing. In Proceedings of the 2017 International Workshop on Crowdsourcing in Software Engineering, Buenos Aires, Argentina, 22–22 May 2017; IEEE Press: Piscataway, NJ, USA, 2017; pp. 30–34.
19. Guzman, E.; Ibrahim, M.; Glinz, M. Prioritizing User Feedback from Twitter: A Survey Report. In Proceedings of the 2017 International Workshop on Crowdsourcing in Software Engineering, Buenos Aires, Argentina, 22–22 May 2017; IEEE Press: Piscataway, NJ, USA, 2017; pp. 21–24.
20. Lakhani, K.; Garvin, D.; Lonstein, E. *Topcoder.com: Developing Software through Crowdsourcing*; Social Science Electronic Publishing: New York, NY, USA, 2010.
21. Mao, K.; Yang, Y.; Li, M.; Harman, M. Pricing crowdsourcing-based software development tasks. In Proceedings of the 2013 International Conference on Software Engineering, San Francisco, CA, USA, 18–26 May 2013; IEEE Press: Piscataway, NJ, USA, 2013; pp. 1205–1208.
22. Archak, N. Money, glory and cheap talk: Analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on Topcoder.com. In Proceedings of the 19th International Conference on World Wide Web, Raleigh, NC, USA, 26–30 April 2010; ACM: New York, NY, USA, 2010; pp. 21–30.
23. Archak, N.; Sundararajan, A. Optimal design of crowdsourcing contests. In Proceedings of the ICIS 2009, Phoenix, AZ, USA, 15–18 December 2009; p. 200.
24. Zhang, H.; Wu, Y.; Wu, W. Analyzing Developer Behavior and Community Structure in Software Crowdsourcing. In *Information Science and Applications*; Lecture Notes in Electrical Engineering; Kim, K., Ed.; Springer: Berlin/Heidelberg, Germany, 2015; Volume 339.
25. Dubey, A.; Abhinav, K.; Taneja, S.; Viridi, G.; Dwarakanath, A.; Kass, A.; Kuriakose, M.S. Dynamics of Software Development Crowdsourcing. In Proceedings of the 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE), Irvine, CA, USA, 2–5 August 2016; pp. 49–58.

26. Dwarakanath, A.; Shrikanth, N.C.; Abhinav, K.; Kass, A. Trustworthiness in enterprise crowdsourcing: A taxonomy & evidence from data. In Proceedings of the 38th International Conference on Software Engineering Companion, Austin, TX, USA, 14–22 May 2016; ACM: New York, NY, USA, 2016; pp. 41–50.
27. Alelyani, T.; Yang, Y. Software crowdsourcing reliability: An empirical study on developers behavior. In Proceedings of the 2nd International Workshop on Software Analytics (SWAN 2016), Seattle, WA, USA, 13 November 2016; ACM: New York, NY, USA, 2016; pp. 36–42.
28. Hu, Z.; Wu, W. A game theoretic model of software crowdsourcing. In Proceedings of the 2014 IEEE 8th International Symposium on Service Oriented System Engineering, Oxford, UK, 7–11 April 2014.
29. Hu, Z.; Wu, W. Game Theoretic Analysis for Offense-Defense Challenges of Algorithm Contests on Topcoder. In Proceedings of the 2015 IEEE Service-Oriented System Engineering, San Francisco Bay, CA, USA, 30 March–3 April 2015; pp. 339–346.
30. Kuang, L.; Zhu, Y.; Li, S.; Yan, X.; Yan, H.; Deng, S. A Privacy Protection Model of Data Publication Based on Game Theory. *Secur. Commun. Netw.* **2018**, *2018*, 3486529. [[CrossRef](#)]
31. Machado, L.; Zanatta, A.; Marczack, S.; Prikladnicki, R. The Good, the Bad and the Ugly: An Onboard Journey in Software Crowdsourcing Competitive Model. In Proceedings of the 2017 IEEE/ACM 4th International Workshop on CrowdSourcing in Software Engineering (CSI-SE), Buenos Aires, Argentina, 22–22 May 2017.
32. Liao, Z.; He, D.; Chen, Z.; Fan, X.; Zhang, Y.; Liu, S. Exploring the Characteristics of Issue-related Behaviors in GitHub Using Visualization Techniques. *IEEE Access* **2018**, *6*, 24003–24015. [[CrossRef](#)]
33. Zhang, Z.; Zhang, J.; Fan, C.; Tang, Y.; Deng, L. KATZLGO: Large-scale Prediction of LncRNA Functions by Using the KATZ Measure Based on Multiple Networks. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2017**. [[CrossRef](#)]



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).