Journal of Information Processing Vol.22 No.2 202–209 (Apr. 2014)

[DOI: 10.2197/ipsjjip.22.202]

Regular Paper

A Division Strategy for Achieving Efficient Crowdsourcing Contest

Huan Jiang[1,a)] Shigeo Matsubara[1,b)]

Received: July 8, 2013, Accepted: January 8, 2014

Abstract: In this paper we construct and analyze a crowdsourcing-based bug detection model in which strategic players select code and compete in bug detection contests. We model the contests as all-pay auctions, and our focus is on addressing the low efficiency problem in bug detection by division strategy. Our study shows that the division strategy can control two features of the bug detection contest, in terms of the expected reward classes and the scales of skill levels, by intentionally assembling players with particular skill distribution in one division. In this way, division strategy is able to determine the players' strategic behaviors on code selection, and thus improve the bug detection efficiency. We analyze the division strategy characterized by skill mixing degree and skill similarity degree and find an explicit correspondence between the division strategy and the bug detection efficiency. Based on our simulation results, we verified that the skill mixing degree, serving as determinant factor of division strategy, controls the trend of the bug detection efficiency, and skill similarity degree plays an important role in indicating the shape of the bug detection efficiency.

Keywords: incentive design, efficient crowdsourcing, division strategy, software bug detection, all-pay auction

1. Introduction

Crowdsourcing is an online, distributed problem-solving and web-based business model that has emerged in recent years [3], and it is now admired as one of the most lucrative paradigm of leveraging the collective intelligence of crowds. This very success is dependent on the diversity of crowds, not only the diversity of opinion and skill, but also the diversity of *role* played by the crowds in crowdsourcing process.

Common crowdsourcing process proceeds in three phases: (i) a task is announced, usually with its requirements, reward and time period; (ii) crowds work effortfully to compete to provide the best solution; (iii) all of the solutions are examined, a subset of solutions are selected, and the corresponding users are granted the rewards. Although the crowd's primary role is as *task solvers* in the second phase, they become more and more important in the solution examination phase in a wide variety of tasks, work as *solution examiners* to help screening or picking out the best solutions. For example, Stack Overflow, which is a Q&A site for programmers, introduced a *voting system* to help valuable pieces of technical knowledge to become more visible [14]. Specifically, users earn rights to examine the contents of posts (questions and answers) of others and vote on the posts if they are considered to be especially useful and appropriate. Upvote on a post helps it to be more visible — appear in the front of the post list. Another example is Threadless, which grants its community members to *score* the designs of t-shirts submitted by other members [4].

[1] Department of Social Informatics, Kyoto University, Kyoto 606–8501, Japan
[a)] jiang@ai.soc.i.kyoto-u.ac.jp
[b)] matsubara@i.kyoto-u.ac.jp

In this paper, we explore the model for crowdsourcing software development process, where the crowdsourcing-based *bug detection* is imported as a solution examination phase. Specifically, we model this process as a two-stage process in which the crowds who selected the same programming problem announced by crowdsourcing center (i) in the submission stage, work as *coders* independently, and submit their unique pieces of program codes to the crowdsourcing center and (ii) in the bug detection stage, are presented all of the codes have been submitted to the center, select one piece of them and work as *bug detectors*. Each piece of code is endowed with reward and has a single chance to be proved incorrect. Hence, reward is granted to the bug detector who becomes the first one to provide the test case which can prove his selected code to be incorrect. For instance, the algorithm competition on TopCoder.com includes *coding stage*, where crowds are presented with the same programming problem and are supposed to submit their own codes within limited time, and *challenge stage*, where each of the users has a chance to challenge the functionality of the codes from others. A successful challenge results in a 50-point reward for challenging user [13].

We are of the opinion that this bug detection process have some strong advantages. Above all, it is a crowdsourcing-based process that can eliminate the false codes and retain the true ones more efficiently and economically than employing a small number of experts to test the large number of codes. Secondly, bug detection by going through others' programs could be beneficial to the programmers since it gives them a chance to learn from others for skill improvement. In contrast, crowds fail to improve in crowdsourcing without such processes [20].

In spite of the above mentioned advantage, the crowdsourcing-based bug detection paradigm has inherent weakness in term of

the distribution of codes that have been selected to be debugged. Because of the various degrees of the codes' qualities, crowds incline to congest on the ones with relatively low qualities that give them high probabilities of successful bug detection, which leaves a substantial part of unchecked solutions and leads to redundant examination on the other part. From the efficiency standpoint, adopting the perspective of a *principal* with the goal to identify maximum number of incorrect codes, we view the uneven distribution of debugged codes as low efficiency.

The goal of this paper is to mitigate the uneven distribution problem to improve the crowdsourcing-based bug detection from an efficiency standpoint. To achieve this goal, we encounter the challenge of balancing the freedom and restriction on code selection. It is noteworthy that crowdsourcing provides the policy of maximum freedom of task selection, which encourages crowds to solve the tasks based on their own preferences and thus becomes indispensable for the crowdsourcing's prosperity. Because of that, the traditional assignment problem [12], which has been well studied to solve the problem of assigning a given set of workers with varying preferences to a given set of jobs in such a way that, maximum efficiency can be achieved, is not appropriate to be used in crowdsourcing situation.

By contrast, *division strategy* is considered to be highly effective. The basic idea of division strategy is to control the crowds' skill distribution in each division by strategically dividing the crowds with different skills into divisions, and restrict code selection in the division, i.e., the crowds can only select from the codes produced by the crowds in the same division. By doing this, we are able to guide the crowds to select from the codes with more appropriate levels of qualities that corresponding to their skill levels. Therefore, the division strategy can be reliable to work out a more uniform distribution of debugged codes and thus an improvement in bug detection efficiency.

We construct the paper as follows. In Section 3, we model the bug detection contest as *all-pay auction*, and rigorously analyze the relationship between strategic code selection behaviors and the offered rewards. In Section 4, we present the basic idea of the division strategy and explore two key features in efficient division strategy. In Section 5, we conduct the simulation which verifies the effectiveness of division strategy on bug detection efficiency. Finally, Section 6 concludes our results, discusses the limitation of our model and suggests potential topics for future research.

2. Related Work

There has recently been work on addressing the efficiency problem of crowdsourcing where the crowdsourcing-based contests are modeled as all-pay auctions, a well-studied mechanism that is frequently employed in the contest literature [2]. To date, most previous research has focused on the crowdsourcing contests with the format that the principal only benefits from the submission with the highest bid (i.e., the solution with the highest quality), with the aim of optimizing the quality of the best submission [1], [7], [11].

Instead of suffering the loss of submissions provided by non-winners, our research takes the full advantage of all-pay auction, with the goal of optimizing the *sum of qualities* for the principal, in connection to bug detection, the total number of debugged codes. Moldovanu and Sela [17] study the contest model with multiple prizes for the sum-of-qualities objective. Another work [18] proposed by them studies both the highest quality submission objective and the sum of qualities objective in a multi-stage contest model. Minor [16] studies the incentive design for contest with heterogeneous players to elicit the maximum of the sum of qualities. In contrast, Cavallo and Jain [6] consider the crowdsourcing from a social planner's perspective that seeks to maximize social welfare.

The empirical research shows the evidence that both monetary reward and non-monetary reward, taking the form of reputation points, provide incentive for crowds to submit high quality solutions [15], [20]. DiPalantino and Vojnovic [9] theoretically demonstrate that qualities of submissions are logarithmically increasing as a function of the offered reward. Those results motivate the *reward allocation design* in crowdsourcing contests for improving efficiency. Particularly, contest with *multiple prizes* becomes an effective way to maximize the sum of effort investment [5], [8], [17], [19]. In addition, *contest architecture design* is also used for improving contest efficiency. A contest architecture specifies a *multi-stage contest* in which players are split among several sub-contests whose winners compete against each others. Optimal structure and reward allocation are studied in a sheer amount of literatures [10], [18].

Our work differs from both of the above methods mainly in the following two aspects. 1) Instead of considering one contest with multiple rewards, our bug detection model is more akin to a collection of separate contests and each contest has one single reward. 2) Instead of allocating multiple prizes to the winners, we indirectly control the rewards of a set of contests using division strategy. A notably relevant work is Ref. [9].

3. Preliminaries

3.1 Bug Detection Model

It is natural to view the competitive nature of the bug detection stage as a *contest*, i.e., crowds who selected the same piece of code for bug detection compete among themselves for the reward. We model the contest as an *all-pay auction*, and consider an highest-bid-wins single-item all-pay auction, in which bidders simultaneously submit sealed bids for an item. All bidders forfeit their bids. The auctioneer awards the item to the bidder with the highest bid, and keeps all the bids.

In the connection to the bug detection contest, the item is the reward, the bids are the actions of bug detection (e.g., constructing test cases), and the sealed value of the bids is the efforts exerted in the bug detection. Specifically, we consider the bug detection stage as a game in which each player, i.e., the bug detector, selects a contest, i.e., a unique piece of code, exerts effort and in each contest the player with the highest effort wins the contest. In the event of a tie, a winner is selected uniformly at random among the highest bidders. Consider there are N players. Associated with each player i is his skill parameter, v_i, where $v_i \in (0, 1]$ is drawn from a non-decreasing distribution function $F(v)$ independently of the skills of the other players. Player i's skill is higher than player j's if $v_i > v_j$.

The players can be naturally partitioned into K ($K \geq 2$) classes, from low skill to high skill, according to the *skill classes* $\vec{\theta} = (\theta_0, \theta_1, \ldots, \theta_K)$, $\theta_0 = 0 < \theta_1 < \cdots < \theta_K = 1$. Thus the k-th class contains the players with k-th lowest skills, $v \in (\theta_{k-1}, \theta_k]$. Let N_k be the number of players in class k, then we must have $\sum_{k=1}^{K} N_k = N$.

Let R denote the reward for first successful bug detection for any of the codes, and the value of the reward is common knowledge. However, simply treating the rewards as the same fails to capture the quality difference of codes written by players with different levels of skills, since it is generally acknowledged that there are variations in individual programming performance. One reasonable assumption is that the codes produced by low-skill coders are those with higher probabilities of incorrectness than those produced by high-skill coders. It is implied that the skill parameter v_i stands for both coding and debugging skills of player i. Based on this assumption, we furthermore endow each piece of code with a *weight coefficient* according to its quality, which is positively correlated with the player's skill. Specifically, the codes produced by player from k-th class are associated with weight $\beta_k \in [0, 1]$. Intuitively, for each piece of code, the associated weight can be viewed as its probability of having bugs, we have $\beta_1 > \beta_2 > \cdots > \beta_K$. The *expected rewards* for successful bug detection for the codes produced by k-th class of players can be calculated as $ER_k = \beta_k \cdot R$. That is, for K classes of codes produced by K classes of players, we have K classes of rewards, $\overrightarrow{ER} = (ER_1, \ldots, ER_K)$, where $ER_1 > ER_2 > \cdots > ER_K$, thus, the bug detection contests are naturally partitioned into K classes, which taking on one of these values as its reward.

3.2 Contest Selection

Proposition 1. (*Proposition 3.1 and 4.1* [9]). *There exist a unique symmetric equilibrium to the bug detection contest game.*

Theorem 1. (*Theorem 4.2* [9]). *In the equilibrium, players select unique code as given in the following.*

1. Skill levels. *Players are partitioned over L skill levels. A skill level l corresponds to the interval of skill values* $[v_{l+1}, v_l)$*, where*

$$F(v_l) = 1 - N_{[1,l]}\left(1 - \frac{ER_l^{\frac{1}{N-1}}}{H_{[1,l]}(\overrightarrow{ER})}\right), \quad \text{for } l = 1, \ldots, L \qquad (1)$$

where $H_{[1,l]}(\overrightarrow{ER}) = \left(\sum_{k=1}^{l} \frac{N_k}{N_{[1,l]}} ER_k^{-\frac{1}{N-1}}\right)^{-1}$ *and* $N_{[1,l]} = \sum_{k=1}^{l} N_l$.

2. Code selection vs. skill level. *A player of skill v selects a particular contest/code of class j with probability* $\pi_j(v)$ *given by*

$$\pi_j(v) = \begin{cases} \dfrac{ER_j^{-\frac{1}{N-1}}}{\sum_{k=1}^{l} N_k ER_k^{-\frac{1}{N-1}}}, & \text{for } j = 1, \ldots, l \\ 0, & \text{for } j = l+1, \ldots, K. \end{cases} \qquad (2)$$

for $v \in [v_{l+1}, v_l)$.

Remark. In item 1, Eq. (1) shows that the intervals of skill levels is determined by the players' skill classes and the number of players of each skill class. It is worthwhile to note that, according to Eq. (1), in the equilibrium, two players in the same skill class could be partitioned into different skill levels, and one skill level could be constituted by the players from more than one skill classes. Item 2 shows that, in the equilibrium, a player of skill level l selects a contest that provides one of l highest rewards with probability given by Eq. (2). Specifically, the player with skill level l selects contest that provides the corresponding expected reward, that is, the l-th highest reward, with the highest probability, and those that provides larger expected rewards are selected with lower probabilities.

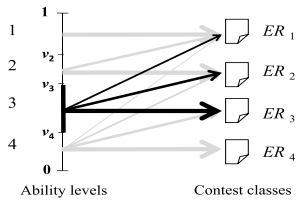


Fig. 1 Contest selection behavior in equilibrium. Players are partitioned into 4 skill levels. A player of skill level 3 selects the contests that provide the 3*rd* highest expected rewards with the largest probability.

Figure 1 provides an illustration of the contest selection behavior for $K = 4$ and $L = 4$. The thickness of the arrows indicates the probability that a player selects that particular contest. A player of skill level 3 will select the contests that offer one of 3 highest expected rewards. Moreover, she selects contests that offer the 3*rd* highest expected reward with the largest probability, and that offer the highest expected reward with the smallest probability.

4. Bug Detection Efficiency Based on Division Strategy

Aimed at identifying the key features of division strategy that determine the bug detection efficiency, we explore division strategies in two dimensions — horizontal and vertical. For the horizontal comparison, we explore the *skill mixing degree*, which varies from none — *separating*, assembles agents with same skill class in the same division, to very high — *mixing*, assembles agents with various skill classes in one division. Based on the example study, we make our first appealing conjecture as follows.

– Skill mixing degree is the most significant feature of the efficient division strategy. Specifically, bug detection efficiency is positively related to the skill mixing degree;

Although the skill mixing degree controls the trend of the bug detection efficiency, such a relationship between skill mixing degree and bug detection efficiency is not a one-to-one correspondence. It inspires us to explore the feature that can differentiate the division strategies wherein the same skill mixing degree leads to different bug detection efficiencies. For this vertical analysis, we propose the concept of skill similarity degree which represents the closeness of players' skills in value. Based on the example study, we make our second appealing conjecture as follows.

– Skill similarity degree plays a subordinate role. Specifically, for the situations wherein the same skill mixing degree leads to different bug detection efficiencies, high skill similarity degree indicates high level of bug detection efficiency.

We arrange this section as follows. First we formally construct the definitions for skill mixing degree, skill similarity degree and bug detection efficiency. Then we conduct an example study. At first, we compare the bug detection efficiencies on different skill mixing degrees, which indicates that the bug detection efficiency increases along with the increase in skill mixing degree. Sec-

ondly, the comparison among the situations wherein the same skill mixing degree leads to different bug detection efficiencies indicates us that high bug detection efficiency also depends on high skill similarity degree. Finally, weighted average of skill mixing degree and skill similarity degree is constructed for evaluating their relative importance.

4.1 Division Strategy

Definition 1. *The skill mixing degree is defined as the sum of the degrees of the skill mixing of all divisions, as follows*

$$M = \sum_{i=1}^{I} M_i \tag{3}$$

where I is the number of divisions, and

$$M_i = \frac{\prod_{k\in\Gamma_i} N_{ik}}{\sum_{k\in\Gamma_i} N_{ik}^2} \tag{4}$$

where $\Gamma_i = \{k : N_{ik} > 0, k \in \{1, \ldots, K\}\}$*, and* N_{ik} *is the number of the players of k-th skill class in division i.*

Definition 2. *The skill similarity degree is defined as the sum of the degrees of the skill similarity of all divisions, as follows*

$$S = \sum_{i=1}^{I} S_i \tag{5}$$

where I is the number of divisions, and

$$S_i = 1 - \sqrt{\frac{1}{N_i - 1}\sum_{k=1}^{N_i-1}(v_k - v_{k+1})^2} \tag{6}$$

where N_i *is the number of the players in division i, and the all players' skills are sorted in non-decreasing order, i.e.,* $v_1 \le v_2 \le \cdots \le v_{N_i}$.

In order to clarify the efficacy of the division strategy, let's first consider the problem of *uneven distribution* of debugged codes in bug detection contests without division strategy, which is reduced to the situation wherein all players are assembled in the unique division. When players of highest skill class and lowest skill class are assembled in the same division, according to Eq. (2), players with highest skills will exclusively select the codes produced by the lowest-skill player which offer the highest expected rewards. Consequently, in order to avoid competing with the players having higher skills, players with lowest skills would probably select the codes with the highest qualities. Moreover, due to their low skills, they can hardly catch the bugs in those codes. This results in low efficiency, from the principal's point of view, although codes produced by low-skill player would be examined, those produced by high-skill players are left unchecked or checked superficially due to the debuggers' limited skills. In addition, from the player's point of view, it leads to significant inequality and harms the benefits gained by low-skill players.

By applying the division strategy, we can mitigate the uneven distribution of the checked codes and the inequality by restricting the players to select the codes with more appropriate levels of rewards. Specifically, in our bug detection model, we divide N players into I divisions, and restrict the contest selection in the divisions, i.e., the players can only select from the codes produced by the players in the same division. By intentionally assembling players with particular skills classes in one division, the principal can control the essential features of the collection of contests in the division, in terms of expected reward classes and the scales of skill levels, which, according to Eqs.(1) and (2), determine the players' strategic behaviors on code selection. For instance, when the players of highest skill class and those of lowest skill class are partitioned into different division, the former are restricted to select the codes produced by the players with higher skill classes instead of the lowest skill class.

In the division, the expected reward class is the most significant determinant of the players' strategic behaviors on code selection, and there's one-to-one correspondence between it and the composition of players' skill classes, in terms of the number of the skill classes and the number of players in each skill class. Hence, the concept of skill mixing degree has been constructed for distinguishing the division strategies varying in compositions of players' skill classes, and becomes the most important factor in our bug detection model.

Moreover, it is worthwhile to note that, the exchange of two players belong to two different divisions can bring different effects on the skill mixing degree and skill similarity degree. Specifically, if the two players in two divisions has similar skill values but belong to different skill classes (since the skill values for players are continuous, two players with similar skill values could be in two adjacent but different skill classes.), the exchange of these two player will cause small similarity degree changes in both divisions, but a relatively large change in mixing degree. It is reasonable based on the definitions given by Eqs. (4) and (6) that skill mixing degree concerns the number of players in different skill classes, but the skill similarity degree concerns the specific value of the players skills. In the same way, we can deduce that if there's a large difference between the skill values of two players in two divisions and belong to different skill classes, the exchange of them will cause relatively small change in mixing degree, but a large change in similarity degree.

4.2 Bug Detection Efficiency

Definition 3. *The bug detection efficiency is defined as the sum of the bug detection efficiency of all divisions, as follows*

$$E = \sum_{i=1}^{I} E_i \tag{7}$$

where E_i *is the efficiency of the i-th division, which is defined as*

$$E_i = \sum_{n=L}^{1}\sum_{m=L}^{n} R \cdot \beta_m \cdot N_{im} \cdot \pi_{L-m+1}(v_{L-n+1}) \int_{v_{L-n+2}}^{v_{L-n+1}} v dv. \tag{8}$$

where

- *R: the reward;*
- β_m*: the weight coefficient endowed to m-th class of codes, representing its probability of having bugs;*
- N_{im}*: the number of players of m-th skill class in division i;*
- $\pi_{L-m+1}(v_{L-n+1})$*: the probability for a player with skill level* $L - n + 1$ *to select a particular code of class* $L - m + 1$*;*
- $[v_{L-n+2}, v_{L-n+1})$*: the interval of skill level* $L - n + 1$.

Remark. The integrals of the skill parameter imply another

Table 1 An example of bug detection efficiency increases with the skill mixing degree of division strategy.

	Division Strategy I		Division Strategy II		Division Strategy III	
	Division 1_1	Division 1_2	Division 2_1	Division 2_2	Division 3_1	Division 3_2
Division Initialization	0.1 R_1 0.2 R_1 0.3 R_1 0.4 R_1	0.5 R_2 0.6 R_2 0.7 R_2 0.8 R_2	0.1 R_1 0.2 R_1 0.3 R_1 0.7 R_2	0.4 R_1 0.5 R_2 0.6 R_2 0.8 R_2	0.1 R_1 0.2 R_1 0.7 R_2 0.8 R_2	0.3 R_1 0.4 R_1 0.5 R_2 0.6 R_2
Skill Mixing Degree (M)	$M_1 = 0.5$		$M_2 = 0.6$		$M_3 = 1$	
Skill Similarity Degree (S)	$S_1 = 1.8$		$S_2 = 1.62$		$S_3 = 1.6$	
Skill Level (L)	$L_{1\_1} = 1$	$L_{1\_2} = 1$	$L_{2\_1} = 2$	$L_{2\_2} = 2$	$L_{3\_1} = 2$	$L_{3\_2} = 2$
	$<v_2, v_1>$=<0,1>	$<v_2,v_1>$=<0,1>	$<v_3, v_2, v_1>$ =<0, 0.38, 1>	$<v_3, v_2, v_1>$ =<0, 0.79, 1>	$<v_3, v_2, v_1>$ =<0, 0.59, 1>	$<v_3, v_2, v_1>$ =<0, 0.59, 1>
Contest Selection						
Bug Detection Efficiency (E)	E_{1_1}=0.8	E_{1_2}=1.04	E_{2_1}=0.68	E_{2_2}=1.24	E_{3_1}=1.32	E_{3_2}=0.84
	E_1=1.84		E_2=1.92		E_1=2.16	

reasonable assumption that the player's skill of coding is proportional to her skill of bug detection. Therefore, the skill with a player can be viewed as her probability of successful bug detection. Then $\pi \cdot v$ can be understood as the probability of one piece code to be selected and successfully proved to be flawed by a player with skill v, and $\beta \cdot N$ is the number of incorrect codes produced by the player in the same division. Hence, when the reward is normalized to 1, the efficiency of bug detection defined above can be understood as the expected number of codes proved to be flawed in all divisions.

4.3 Example Study: Key Factors in Efficient Division Strategy

In **Table 1**, we consider a two-division situation wherein eight players who are endowed with skills $v_1, v_2, \cdots, v_8 = 0.1, 0.2, \cdots, 0.8$, belong to two skill classes, $(0, 0.4]$ and $(0.4, 0.8]$. The two classes of codes produced by low-skill players and high-skill players are, without loss of generality, endowed with specific values, $R_1 = 0.8$ and $R_2 = 0.4$, respectively. We initialize the two divisions with equal number of players. In Division Strategy I, two classes of players are completely separated into two divisions, which leads to the minimum skill mixing degree $M_1 = 0.5$. In the equilibrium, players in both divisions are partitioned into only one skill level ($L_{1\_1} = L_{1\_2} = 1$). Therefore, in Division 1_1, players have equal probability for selecting among the four high-reward contests.

Consider the *common case* in which 1) players in high-skill level select contests with high rewards and players in low-skill level select those provide low rewards, 2) the "congestion problem" — more than one players compete on the same piece of code — is minimized. Thus, in Division 1_1 no players is supposed to compete on any of the contests, and each piece of code is debugged by a unique player. By contrast, the situation is different in division 2_1 brought by Division Strategy II, where players are partitioned into two skill levels in the equilibrium. There's one player ($v = 0.7$) in the high-skill level, who is also the only one can select from three high-reward contests. The other three players of low-skill level have to compete for the only contest with low reward R_2. Player with the highest skill ($v = 0.3$) wins the reward.

The horizontal comparison among the outcomes under different division strategies demonstrates our first conjecture, that bug detection efficiency is positively related to the skill mixing degree. Specifically,skill mixing degrees of Division Strategy I, II, III are $M_1 = 0.5$, $M_2 = 0.6$, $M_3 = 1$, and the corresponding bug detection efficiency are $E_1 = 1.84$, $E_2 = 1.92$, $E_3 = 2.16$.

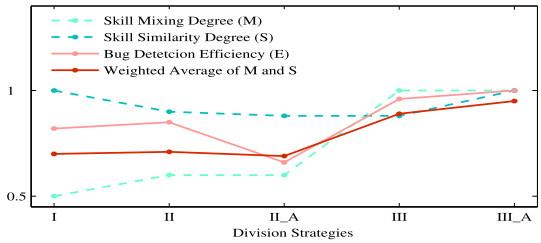
Table 2 Efficiency increases with the skill similarity degree.

	Division Strategy II_A		Division Strategy III_A	
M	$M_{2\_A} = 0.6$		$M_{3\_A} = 1$	
L	$<v_3, v_2, v_1>$ =<0, 0.38, 1>	$<v_3, v_2, v_1>$ =<0, 0.79, 1>	$<v_3, v_2, v_1>$ =<0, 0.59, 1>	$<v_3, v_2, v_1>$ =<0, 0.59, 1>
Contest Selection				
S	$S_{2\_A} = 1.6$		$S_{3\_A} = 1.8$	
E	$E_{2\_A\_1}$=0.76	$E_{2\_A\_2}$=0.72	$E_{3\_A\_1}$=0.76	$E_{3\_A\_2}$=1.48
	E_{2_A}=1.48		E_{3_A}=2.24	

In **Table 2**, we re-initialize the player allocation in Division Strategy II. By exchanging two players with skills $v = 0.7$ and $v = 0.8$, the bug detection efficiency decreases to $E_{2\_A} = 1.48$ ($< E_2 = 1.92$), without any change to the skill mixing degree ($M_{2\_A} = M_2 = 0.6$). By contrast, the change of player allocation in Division Strategy III leads to contrary outcome. By exchanging the high-skill-class players in two divisions, the bug detection efficiency increases to $E_{3\_A} = 2.24$ ($> E_3 = 2.16$), with the same skill mixing degree ($M_{3\_A} = M_3 = 1$).

The one-to-many correspondence between skill mixing degree and bug detection efficiency inspires us to do the vertical comparison by checking the skill similarity degree of division strategies with the same skill mixing degree. For the Division Strategy II and II_A, with the same skill mixing degree ($M_{2\_A} = M_2 = 0.6$), II_A with lower skill similarity degree ($S_{2\_A} = 1.6 < S_2 = 1.62$) works out lower bug detection efficiency ($E_{2\_A} = 1.48 < E_2 = 1.92$). For the Division Strategy III and III_A, with the same skill mixing degree equals to 1, the increase in skill similarity degree of III_A ($S_{3\_A} = 1.8 > S_3 = 1.6$) leads to a higher bug detection efficiency ($E_{3\_A} = 2.24 > E_3 = 2.16$). This vertical comparison demonstrates our second conjecture that for the situations wherein the same skill mixing degree leads to different bug detection efficiencies, high skill similarity degree indicates high level of bug detection efficiency.

Based on the above analysis, we go one step further by incorporating the relative importance relation of skill mixing degree and



	Skill Mixing Degree (M)	Skill Similarity Degree (S)	Bug Detection Efficiency (E)	Weighted Average
I	0.50	1.00	0.82	0.70
II	0.60	0.90	0.85	0.71
II_A	0.60	0.88	0.66	0.69
III	1.00	0.88	0.96	0.89
III_A	1.00	1.00	1.00	0.95

Fig. 2 Relative importance of two key factors of efficient division strategy. Shape of weighted average ($(M + 0.9S)/2$) over skill mixing degree and skill similarity degree is consistent with that of bug detection efficiency (E).

skill similarity degree. Using weighted average to combine these two determinant factors allows to consider their relative importance in division strategy for bug detection efficiency. **Figure 2** visualizes the shape of the value of the weighted averages over the skill mixing degrees and skill similarity degrees ($(M + 0.9S)/2$) given in the example. By weighting skill mixing degree and skill similarity degree with coefficients 1 and 0.9 respectively, the average weight of skill mixing degree and skill similarity degree is demonstrated to be consistent with the shape of bug detection efficiency. Although, 0.9 is just an instance for the weight on skill similarity degree, one can easily find that once the coefficient given to skill similarity degree is equal or larger than the one of skill mixing degree, the weighted average becomes different from the bug detection efficiency. The values of weight demostrate that skill mixing degree plays a more important role than skill similarity in the bug detecttion efficiency.

5. Simulation and Analysis

In this section, we experimentally evaluate the performance of the division strategies with different skill mixing degrees and skill similarity degrees. First, we randomly generate N players with different skills. Second, we divide them into I divisions in the way that it initials the division strategy with the lowest skill mixing degree. Without loss of generality, we suppose each division contains the same number of players, i.e., N/I. Based on our model, bug detection efficiency of each division can be computed. Third, by strategically changing the allocation of players, we develop a new division strategy with higher skill mixing degree. By repeatedly conducting the second and the third steps, we can then investigate the conjectures given in the above section.

5.1 Setting

We explore the above two features in three environments.

Linear skill distribution function. Suppose the skills of players are uniformly distributed on the unit interval $[0, 1]$, we assume the skill distribution function as linear, i.e., $F_{linear}(v) = v$, $v \in [0, 1]$.

Concave skill distribution function. In the situation where the contests attract a lot of players with low skills rather than high skill — for example, a relatively new crowdsourcing platform or the early stages in the multi-stage sequential-elimination contests [10] — we assume the skill distribution function as concave function which is a normal distribution with mean μ and variance $\sigma^2 > 0$, i.e., $F_{concave}(v) = \Phi((v - \mu)/\sigma)$, where $\Phi(x) = \frac{1}{\sqrt{2x}} \int_{-\infty}^{x} e^{-(t^2/2)} dt$.

Convex skill distribution function. By contrast, in the situation where the contest attracts more high-skill players than low-skill players — for example, the crowdsourcing contests with minimum skill requirement — we assume the skill distribution function as convex function which, without loss of generality, can be the following power function: $F_{convex}(v) = v^{\alpha}$, $\alpha > 1$.

Table 3 Summary of parameters.

Parameter	Symbol	Value
No. of Players	N	200
Skill of Players	v	$(0, 1]$
No. of Skill Classes	K	4
No. of Divisions	I	4
Thresholds of Skill Classes	$\vec{\theta}$	$(0, 0.25, 0.5, 0.75, 1)$
Weight Coefficients	$\vec{\beta}$	$(0.8, 0.6, 0.4, 0.2)$

Parameters used in simulation are presented in **Table 3**. 200 players of 4 skill classes are equally divided into 4 divisions. As the weight coefficients show, we don't guarantee the high-quality codes have no bugs, and not all of the low-quality codes are incorrect. We normalize the reward to 1, and we also normalize the simulation results, including bug detection efficiency, skill mixing degree and skill similarity degree in order to compare the effect of different division strategies.

5.2 Results and Discussion

Bug detection efficiency of division strategies with different skill mixing degree are investigated under three kinds of skill distributions. Besides the common case, we also simulate the code selection behavior strictly in accordance with Theorem 1.

First, we can easily confirm the conjecture that the bug detection efficiency increases with the skill mixing degree, for all of the three types of skill distributions. In **Fig. 3**, (a.1) presents the linear skill distribution with mean skill value 0.7, and (a.2) shows the mean efficiency of bug detection versus the skill mixing degree. The linear regression on the bug detection efficiencies indicates an increasing trend for bug detection efficiency through the whole interval of skill mixing degree. **Figure 4** (a.2) and **Fig. 5** (a.2) lead to the same conclusion under both concave skill distribution and convex skill distribution. It is noticeable that, the increase in bug detection efficiency under concave skill distribution situation is particularly remarkable comparing to the situations of linear and convex skill distributions. Specifically, for the concave skill distribution situation, Fig. 4 (a.2) shows the increase in the efficiency is about 0.2 (from 0.6 to 0.8), which is larger than those of the situations for linear skill distributions (Fig. 3 (a.2)) and convex skill distributions (Fig. 5 (a.2)), wherein the increases in the efficiency is about 0.1 (from 0.8 to 0.9, 0.6 to 0.7, respectively). The reason is easy to understand that for the concave skill distribution, with a low mean skill equaling to 0.2782, the number of potential bugs in all codes is higher than that under both linear and convex skill distribution. In Fig. 3 (b.1), Fig. 4 (b.1) and Fig. 5 (b.1) we investigate the mean efficiency versus the skill mixing degree

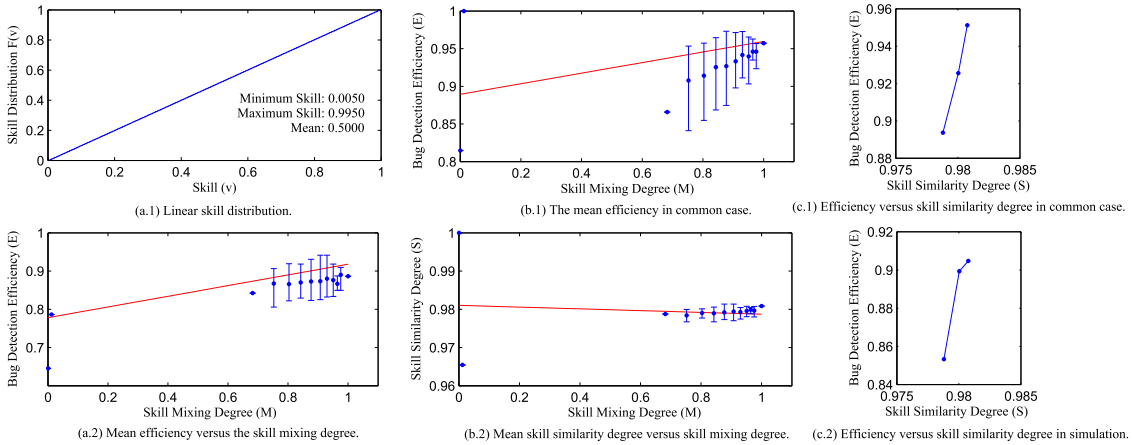


Fig. 3 Linear skill distribution. Bug detection efficiency increases with the skill mixing degree, and shaped as skill similarity degree. The red lines are linear regressions.

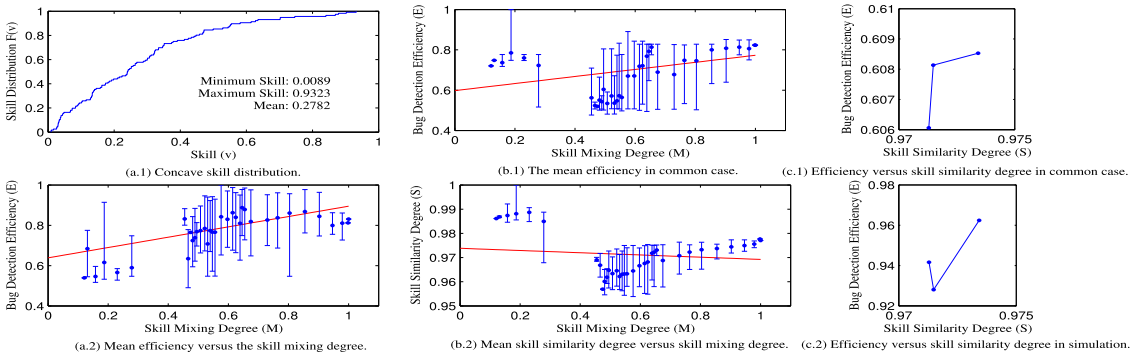


Fig. 4 Concave skill distribution. Bug detection efficiency increases with the skill mixing degree, and shaped as skill similarity degree. The red lines are linear regressions.

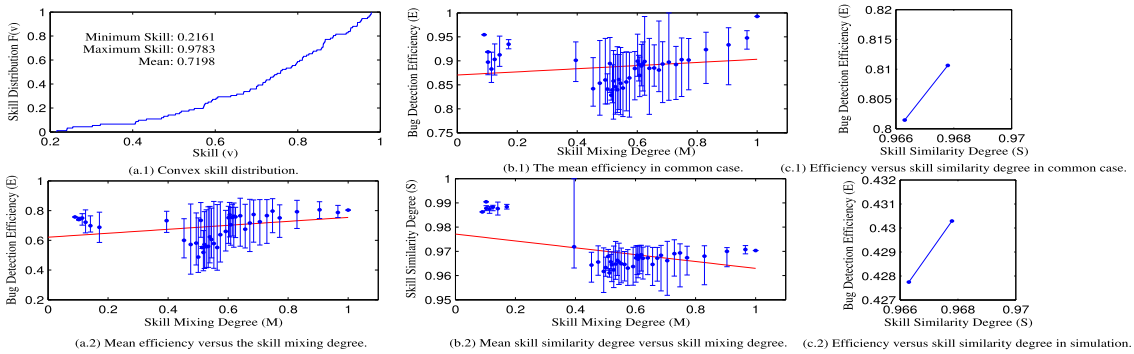


Fig. 5 Convex skill distribution. Bug detection efficiency increases with the skill mixing degree, and shaped as skill similarity degree. The red lines are linear regressions.

in common case. The conjecture that bug detection efficiency is positively related to the skill mixing degree is also hold under three types of skill distributions in this case.

Secondly, we illustrate the mean skill similarity degree versus the skill mixing degree in Figs. 3–5 (b.2). As we initialize the divisions with the lowest skill mixing degree at the very beginning by allocating the players of the same skill class in the same division, the skill similarity degree is set to be high at first. Then by exchanging a small number of players in different skill classes among divisions, the skill similarity degree decreases dramatically along with a slight increase in skill mixing degree. As the exchanging process continues, the numbers of the players in different skill classes become closer, which makes the mean of skill similarity degree rebound gradually. Comparing to the skill similarity degree, the corresponding bug detection efficiency in the common case shapes up in a consistent way. This consistence confirms our conjecture that for the situations wherein the same skill mixing degree leads to different bug detection efficiencies, high skill similarity degree indicates high level of bug detection efficiency.

Furthermore, we present the bug detection efficiency versus skill similarity degree under the same skill mixing degree (by simply choosing the median value of efficiency for illustration) in Fig. 3 (c.1)(c.2), Fig. 4 (c.1)(c.2) and Fig. 5 (c.1)(c.2). They show that in common case, as we conjectured, the efficiency increases with the skill similarity degree. However, it is unlikely that this conjecture holds in the simulation which may due to the random selection on contests. Finally, contrary to the bug detection ef-

ficiency, the linear regression lines on the mean skill similarity degree versus skill mixing degree (Fig. 3 (b.2), Fig. 4 (b.2) and Fig. 5 (b.2)) show that the skill similarity degree decreases along with the increase in skill mixing degree. However, this does not reverse the increasing trend of bug detection efficiency, which indicates that skill mixing degree controls the trend, and skill similarity degree indicates the shape.

6. Conclusion

In this paper we have presented and analyzed a crowdsourcing-based bug detection model in which strategic players select code and compete in bug detection contests. Our focus is on addressing the low efficiency problem in bug detection. Our study shows that by intentionally assembling players with particular skill distribution in one division and limiting the code selection and bug detection contests in the division, the division strategy, to some extent, can control the essential features of the contests, in terms of expected reward classes and the scales of skill levels, which determine the players' strategic behaviors on code selection. By exploring key factors of division strategy, we conclude that skill mixing degree, serving as determinant factor, controls the trend of the bug detection efficiency, specifically, high degree of skill mixing leads to high level of bug detection efficiency, and skill similarity degree plays an important role in indicating the shape of the bug detection efficiency.

Nonetheless, we have noted that although skill distribution as skill mixing degree and skill similarity degree is the main determinant of player strategic behavior on code selection, it is not the only factor that can influence the bug detection efficiency. Future work could consider the impact of the size and the number of the divisions, and how they might affect the bug detection efficiency. In addition, one limitation on the assignment of codes' rewards in the proposed model also could be considered in future research. Aiming at finding as many incorrect codes as possible, we construct expected rewards from the player's point of view to differentiate the qualities of codes and apply the relationship between reward and player's strategic behavior to encourage more appropriate code selection behavior. However, for other situations wherein the principal requires guarantee of high quality on codes produced by high-skill agents, the rewards assigned to the codes produced by high-skill players should be higher than those provided by low-skill players.

Acknowledgments This research was partially supported by a Grant-in-Aid for Scientific Research (S) (24220002, 2012-2016) from Japan Society for the Promotion of Science (JSPS).

References

[1] Archak, N. and Sundararajan, A.: Optimal Design of Crowdsourcing Contests, *Proc. International Conference on Information Systems* (*ICIS*) (2009).
[2] Baye, M.R., Kovenock, D. and De Vries, C.G.: The all-pay auction with complete information, *Economic Theory*, Vol.8, No.2, pp.291–305 (1996).
[3] Brabham, D.C.: Crowdsourcing as a model for problem solving an introduction and cases, *Convergence: The International Journal of Research into New Media Technologies*, Vol.14, No.1, pp.75–90 (2008).
[4] Brabham, D.C.: Moving the crowd at Threadless: Motivations for participation in a crowdsourcing application, *Information, Communication & Society*, Vol.13, No.8, pp.1122–1145 (2010).
[5] Cason, T.N., Masters, W.A. and Sheremeta, R.M.: Entry into winner-take-all and proportional-prize contests: An experimental study, *Journal of Public Economics*, Vol.94, No.9, pp.604–611 (2010).
[6] Cavallo, R. and Jain, S.: Efficient crowdsourcing contests, *Proc. 11th International Conference on Autonomous Agents and Multiagent Systems*, Vol.2, pp.677–686 (2012).
[7] Chawla, S., Hartline, J.D. and Sivan, B.: Optimal crowdsourcing contests, *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, *SODA '12*, pp.856–868 (2012).
[8] Clark, D.J. and Riis, C.: Competition over more than one prize, *The American Economic Review*, Vol.88, No.1, pp.276–289 (1998).
[9] DiPalantino, D. and Vojnovic, M.: Crowdsourcing and all-pay auctions, *Proc. 10th ACM Conference on Electronic Commerce*, *EC '09*, pp.119–128 (2009).
[10] Fu, Q. and Lu, J.: The optimal multi-stage contest, *Economic Theory*, Vol.51, No.2, pp.351–382 (2012).
[11] Ghosh, A. and McAfee, P.: Crowdsourcing with endogenous entry, *Proc. 21st International Conference on World Wide Web*, *WWW '12*, pp.999–1008 (2012).
[12] Kuhn, H.W.: The Hungarian method for the assignment problem, *Naval Research Logistics Quarterly*, Vol.2, No.1-2, pp.83–97 (1955).
[13] Lakhani, K., Garvin, D. and Lonstein, E.: Topcoder (a): Developing software through crowdsourcing, *Harvard Business School General Management Unit Case*, No.610-032 (2010).
[14] Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G. and Hartmann, B.: Design lessons from the fastest q&a site in the west, *Proc. SIGCHI Conference on Human Factors in Computing Systems*, *CHI '11*, pp.2857–2866 (2011).
[15] Mason, W. and Watts, D.J.: Financial incentives and the "performance of crowds", *SIGKDD Explor. Newsl.*, Vol.11, No.2, pp.100–108 (2010).
[16] Minor, D.B.: Increasing Effort Through Softening Incentives in Contests, Technical Report, Working Paper, University of California, Berkeley (2011).
[17] Moldovanu, B. and Sela, A.: The optimal allocation of prizes in contests, *American Economic Review*, pp.542–558 (2001).
[18] Moldovanu, B. and Sela, A.: Contest architecture, *Journal of Economic Theory*, Vol.126, No.1, pp.70–96 (2006).
[19] Szymanski, S. and Valletti, T.M.: Incentive effects of second prizes, *European Journal of Political Economy*, Vol.21, No.2, pp.467–481 (2005).
[20] Yang, J., Adamic, L.A. and Ackerman, M.S.: Crowdsourcing and knowledge sharing: Strategic user behavior on taskcn, *Proc. 9th ACM Conference on Electronic Commerce*, pp.246–255 (2008).

Huan Jiang was born in Chongqing, China, in 1984. She graduated in computer science from Southwest University, Chongqing, China, in 2006, and received her master degree in computer science from Southwest University, Chongqing, China, in 2009. She is now a doctoral student in Department of Social Informatics, Kyoto University, Japan.

Shigeo Matsubara is an associate professor of Department of Social Informatics, Kyoto University. From 1992 to 2006, he was a research scientist of NTT Communication Science Laboratories, NTT. He received his Ph.D. degree in Informatics from Kyoto University. During 2002–2003, he was a visiting researcher at University of California, Berkeley. He was also an advisor of NICT Language Grid project from 2006 to 2007. His research focuses on multiagent systems and information economics. He is a member of JSAI, JSSST, IPSJ and IEICE.