Improving Satisfaction in Crowdsourcing Platforms

Griffith Samore
Department of Computer Science
Hendrix College
Conway, AR 72032
gsamore99@gmail.com

Jonah Bates
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623
jxb8517@rit.edu

Xiao Chen
Department of Computer Science
Texas State University
San Marcos, TX 78666
xc10@txstate.edu

***Abstract*—Crowdsourcing platforms have gained popularity in recent years. They allow requesters to quickly find workers to complete small tasks and workers to undertake pieces of work for self-fulfillment. The current crowdsourcing models do the task assignment either from the requester's side (server assigned task mode) or the worker's side (worker selected task mode). The satisfaction of both sides is not fully considered. Furthermore, there is a lack of tools to help them make decisions based on complex information and their preferences. Therefore, in this paper, we propose a new crowdsourcing platform that takes the satisfaction of both the requesters and workers into account so as to improve the quality of the platform. We adopt Analytic Hierarchy Process (AHP) to automatically generate preference lists for both parties that best reflect their interests. We propose a stable matching (SM) algorithm to pair the workers and tasks according to their preference lists. Simulation results show that our platform has higher satisfaction scores than the existing ones and the one that uses random assignment.**

***Index Terms*—analytic hierarchy process, crowdsourcing, preference list, stable matching, task assignment**

I. Introduction

Crowdsourcing [4] has gained popularity in recent years because it allows requesters to find a crowd of workers to work on small tasks that an individual or an organization cannot easily do and it let workers obtain tasks for money and self-fulfillment. There are three basic components in crowdsourcing: *requesters* who publish tasks on a platform, *workers* who carry out tasks, and a *platform* that permits a one-to-one task to worker pairing. There are various crowdsourcing platforms in our daily lives. For example, Amazon Mechanical Turk [2] offers a place for individuals and businesses to outsource jobs such as data validation and survey participation to workers. Fiverr [5] allows individuals to recruit workers to perform small odd jobs - ranging from graphic design to music composition. And Chegg Tutors [3] makes it possible for people to quickly hire online tutors.

In the crowdsourcing platforms, there are two task assignment modes: *worker selected task mode* and *server assigned task mode* [13]. In the worker selected task mode [10], [15], the server publishes requesters' tasks and it is the workers' responsibility to choose any tasks that they are interested in. One drawback of this mode is that it might neglect jobs that might not seem appealing, resulting in tasks never being completed. On the other hand, in the server assigned task mode [9], [13], [19], the tasks are assigned to workers by the server on behalf of the requesters to optimize some objective functions such as to minimize the cost of autonomous car sharing [12], or to maximize the number of location-related tasks assigned [14], etc. The problem of this mode is that it may lead to low-quality contributions if there are unqualified participants. Right now, these two modes are totally separated, which causes the task assignment to favor one side over the other. The satisfaction of both sides is not fully considered.

Furthermore, in a crowdsourcing platform, the requesters can see many workers with different *attributes* such as their availability, previous reviews, desired pay rate, etc. These attributes have different units and are on different scales. They have different importance to individual requesters. Some attribute, e.g., previous reviews, counts positively (the higher, the better) towards the requester's preference and some of them, e.g., availability and pay rate, count negatively (the higher, the worse) towards the requester's preference. The requesters will be at a loss to pick the best workers for their tasks with so much overwhelming information. Similarly, it would be hard for the workers to find their best tasks out of a large number of tasks with multiple attributes such as the reward of the task, the level of specialization required, the time required to complete the task, etc. Therefore, it is necessary for the platform to provide a facility to help the requesters and workers to make their best choices so that their satisfaction can be improved and consequently the platform will look more appealing to them.

In order to address the aforementioned issues, in this paper we propose a new crowdsourcing model presented in Fig. 1. On our platform, the *attributes* of each side are visible to the other side. Some of the workers' attributes (e.g., availability, desired pay rate) are entered into the platform when they register with the system and some (e.g., previous reviews) are available when people write reviews on the platform after the tasks are done. The attributes of the tasks are input by the requesters when they post the tasks. To help the requesters and workers to process the information and make the best choices, we adopt the *analytic hierarchy process* (AHP) [1] method, which is a structured technique for making good decisions after organizing and analyzing complex criteria using mathematics and personal preference. AHP takes the attributes and the relative importance of these attributes (*weights*) deemed by the decision maker as inputs and generates a preference list ordered from the favorite to the least preferred for the decision maker. A requester

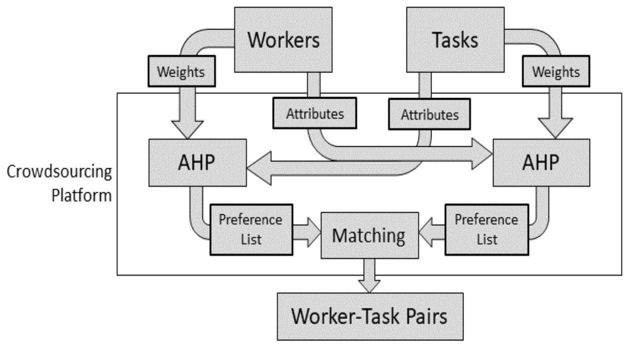


Fig. 1. Our Crowdsourcing Platform

will get a preference list of the workers and a worker will get a preference list of the tasks. Then the platform will use an algorithm called *Stable Matching* (SM) derived from the stable marriage approach [11] to pair the tasks and the workers according to their preference lists. Stable matching can greatly improve the satisfaction of the requesters and workers because it leads to a state where no worker prefers another task than his matched one and no requester prefers another worker than his assigned one.

The differences of our work from others and the key contributions of our work are as follows:

- We propose a crowdsourcing platform model that considers the satisfaction of both the requesters and the workers so as to improve the quality of the platform.
- We use Analytic Hierarchy Process (AHP) to help generate preference lists for both the requesters and workers to best reflect their wishes.
- We propose a stable matching (SM) algorithm to pair the workers and the tasks based on their preference lists generated by AHP.
- Simulations are conducted to compare the satisfaction of both parties using the methods on our proposed platform with that of the existing ones as well as that of a naive random assignment of tasks and workers.

The rest of the paper is organized as follows: Section II references the related works. Section III defines the problem. Section IV introduces the AHP method. Sections V and VI explain how to use AHP to generate a requester's preference list and a worker's preference list, respectively. Section VII matches the workers and tasks using the stable matching (SM) algorithm. Section VIII describes the simulations we have conducted, and the conclusion is in Section IX.

II. Related Works

Crowdsourcing covers a wide spectrum. In terms of task assignment, there are two modes: worker selected task mode and server assigned task mode [13]. In the worker selected task mode [10], [15], the server publishes the tasks and it is the workers' call to choose any tasks they are interested in. One drawback of this method is that the server does not have control over the allocation of the tasks. This may lead to some tasks never been assigned. Differently, in the server assigned task mode [9], [13], [19], the task is totally assigned by the server according to certain rules. This method has a global picture of the tasks so it can achieve the global optimum in terms of some objective functions. But it does not consider much about the quality of the contributions. There are several server assigned task algorithms in some specific applications. For example, in the car sharing industry, [12] provides a greedy method that assigns a passenger request to its geometrically nearest taxi. In the spatial-crowdsourcing environment, [9], [19] assign tasks to workers according to their positions and then the workers will physically move to the specified locations to conduct tasks.

Right now, the worker selected task mode and the server assigned task mode are separated in the existing crowdsourcing platforms. To the best of our knowledge, the research that considers the combination of both modes is our previous work [8]. In that paper, on the requester's side, we put forward a formula derived from Beysian inference to rank workers by considering two factors: previous reviews and the prices the workers ask. On the worker's side, we let each worker decide his task preference list manually. In this paper, we extend, unify, and automate the ranking process on both sides using AHP [1]. AHP was developed by Thomas L. Saaty in the 1970s and is used around the world in a wide variety of decision situations. By using AHP, our proposed platform can generate the preference lists for both sides that reflect their best interests after processing multiple complex criteria. The algorithm we use to pair the workers with the tasks is the stable matching (SM) algorithm derived from the solution to the Stable Marriage Problem (SMP) [16]. SMP aims to find a stable matching between two equally-sized sets of elements (i.e., men and women) given complete preference orders of each man and woman. Stability requires that a matched man and a matched women will not prefer each other over their existing partners.

III. Problem Definition

On our proposed platform, we assume that there are a set of small tasks $T = \{t_1, t_2, \cdots, t_n\}$ posted by requesters and a set of workers $W = \{w_1, w_2, \cdots, w_n\}$ looking for tasks to make some profit. When posting a task, a requester gives the attributes of a task such as its level of specialization, reward, urgency, etc. Similarly, a worker provides his information such as availability, desired pay rate, etc., when he registers with the platform. The platform can also obtain some information such as reviews from the requesters after jobs are done. We assume that the platform uses an evaluation system on the workers in the form of $X\%$ positive out of Y reviews. Different crowdsourcing websites [3], [5] may use different formats to evaluate the workers, but they can be converted to the format we adopt here.

Our overall goal in the platform is to improve the satisfaction of both the requesters and the workers to make the platform more attractive to them. To do that, our first objective is to find a structured method to assist both parties to process complex information and produce preference lists in their best interests. Our next target is to design an algorithm to pair the workers with the tasks stably without each looking for other choice according to their preferences to improve their satisfaction. We not only cover the case that the numbers of workers and tasks are the same but also extend the algorithm to the case when their sizes are different.

IV. The AHP Method

In this section, we introduce the AHP method [18] that is adopted by our crowdsourcing platform to generate the requester's preference list $(w_1, w_2, \cdots, w_n)$ and the worker's preference list $(t_1, t_2, \cdots, t_n)$. Since ranking the workers and the tasks follow the same process, we call workers and tasks the *choices*. So the AHP method is used to rank the available choices $c_1, c_2, \cdots, c_n$ (either workers or tasks) based on multiple attributes $\alpha_1, \alpha_2, \cdots, \alpha_m$ a user considers. The AHP method is composed of the following steps.

Step 1, the user creates a *Pairwise Comparison Matrix* (PCM) shown in Fig. 2, where each element α_{ij} is the relative importance of attribute α_i over attribute α_j deemed by the user. In AHP, a user uses a scale of 1-9 to rate the relative importance of the attributes. For example, if a user sets α_{ij} to 2, this means that the user thinks attribute α_i is two times as important as α_j. Then obviously, attribute α_j is half as important as attribute α_i. That is, $\alpha_{ji} = \frac{1}{2}$. In the matrix, all the elements on the diagonal are reduced to 1 because attribute α_i is as important as itself.

$$\begin{array}{c|c} Attribute\backslash Attribute & \begin{matrix} \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_m \end{matrix} \\ \begin{matrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_m \end{matrix} & \begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & \cdots & \alpha_{1m} \\ 1/\alpha_{12} & 1 & \alpha_{23} & \cdots & \alpha_{2m} \\ 1/\alpha_{13} & 1/\alpha_{23} & 1 & \cdots & \alpha_{3m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1/\alpha_{1m} & 1/\alpha_{2m} & 1/\alpha_{3m} & \cdots & 1 \end{pmatrix} \end{array}$$

Fig. 2. The Pairwise Comparison Matrix (PCM)

Step 2, we need to check the consistency of PCM. When the user fills out the PCM, the resulting matrix may not be consistent. *Matrix consistency* is defined as follows: a positive $m \times m$ matrix A is consistent if $a_{ij}a_{jk} = a_{ik}$, for $i, j, k = 1, \cdots, m$. In a PCM, if the user defines attribute α_i as being twice as important as α_j, and α_j to be three times as important as α_k, then α_i should be six times as important as α_k. But the user may not observe this relationship while populating the PCM. If the PCM is not consistent, we adopt the linearization technique in [6] to convert it to its closest consistent matrix. Due to space limitation, we do not expand the method here. The detailed theory and the matlab code to do the conversion can be found in [6].

Step 3, we calculate the weight λ_i of each attribute α_i. We put these weights into a vector $\Lambda = (\lambda_1, \lambda_2, \cdots, \lambda_m)$. Each element λ_i is computed as:

$$\lambda_i = \frac{(\prod_{j=1}^{m} \alpha_{ij})^{1/m}}{\sum_{i=1}^{m}(\prod_{j=1}^{m} \alpha_{ij})^{1/m}} \tag{1}$$

Obviously, $\sum_{i=1}^{m} = 1$.

Step 4, we put all the values of the attributes for all the choices into matrix X as presented in Fig. 3, where each element χ_{ij} is the value of attribute α_j of choice c_i.

$$\begin{array}{c|c} Choice\backslash Attribute & \begin{matrix} \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_m \end{matrix} \\ \begin{matrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{matrix} & \begin{pmatrix} \chi_{11} & \chi_{12} & \chi_{13} & \cdots & \chi_{1m} \\ \chi_{21} & \chi_{22} & \chi_{23} & \cdots & \chi_{2m} \\ \chi_{31} & \chi_{32} & \chi_{33} & \cdots & \chi_{3m} \\ \vdots & \vdots & \vdots & & \vdots \\ \chi_{n1} & \chi_{n2} & \chi_{n3} & \cdots & \chi_{nm} \end{pmatrix} \end{array}$$

Fig. 3. Matrix X

Step 5, we transform matrix X. This step is important because the attributes we consider can be on different scales and can contribution positively or negatively to our preference. For some attributes, it is the higher the value, the better while for others, the lower the value, the better. So we need to do a transformation on each element in matrix X as follows. For all the values in column α_j, if they contribute positively to our preference, then for each value χ_{ij} in the column of α_j, we change it to $\chi_{ij}/max_{i=1}^{m}(\chi_{ij})$. If they contribute negatively to our preference, then for each value χ_{ij} in column α_j, we change it to $min_{i=1}^{m}(\chi_{ij})/\chi_{ij}$. After this transformation, the values in different attributes are normalized and contribute positively to our preference.

Step 6, we calculate the final scores of all the choices and sort them to obtain the preference list. The final scores of all the choices are put in an $n \times 1$ matrix S which is calculated as follows:

$$S_{n\times 1} = X_{n\times m}\Lambda_{m\times 1}$$

Finally, we can obtain our preference list by sorting the choices according to their scores in a non-increasing order.

V. Requester's Preference List

In this section, we explain the concrete procedure to generate a requester's preference list using AHP.

As a toy example, let us assume that a requester is looking for a worker to complete a job. To rank the workers, he considers three attributes: the availability of the worker (α_1), worker's previous reviews (α_2), and the worker's desired pay rate (α_3). Based on his opinion of the importance of the three attributes, his PCM is set as follows:

$$\begin{array}{c|c} Attribute\backslash Attribute & \begin{matrix} \alpha_1 & \alpha_2 & \alpha_3 \end{matrix} \\ \begin{matrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{matrix} & \begin{pmatrix} 1 & 2 & 1 \\ \frac{1}{2} & 1 & 3 \\ 1 & \frac{1}{3} & 1 \end{pmatrix} \end{array}$$

After the consistency checking, this PCM is not consistent - since α_1 is 2 times as important as α_2 and α_2 is 3 times as important as α_3, α_2 should be 6 times as important as α_3. But α_{13} is labeled 1. We apply the method in [6] to convert the PCM into its closest consistent matrix. This yields:

$$\begin{array}{c c} Attribute \backslash Attribute & \begin{array}{ccc} \alpha_1 & \alpha_2 & \alpha_3 \end{array} \\ \begin{array}{c} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{array} & \begin{pmatrix} 1 & 1.1006 & 1.8171 \\ 0.9086 & 1 & 1.6510 \\ 0.5503 & 0.6057 & 1 \end{pmatrix} \end{array}$$

According to Formula (1), we obtain the weights of the three attributes as: $\Lambda = (0.41, 0.37, 0.22)$. Now suppose we have three workers to choose from. Their values in the three attributes are presented in the following matrix X:

$$\begin{array}{c c} Choice \backslash Attribute & \begin{array}{ccc} \alpha_1 & \alpha_2 & \alpha_3 \end{array} \\ \begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array} & \begin{pmatrix} 1 & 2600 & 15 \\ 3 & 2100 & 10 \\ 2 & 2400 & 12 \end{pmatrix} \end{array}$$

In matrix X, column α_1 lists in how many days each worker is available and column α_3 shows each worker's desired pay rate per hour. These numbers are provided by the workers when they register on the platform. Column α_2 displays each worker's review score. This score is calculated based on a worker's previous reviews using the method in our previous work [8]. The main idea of the method is as follows. Suppose two workers A and B get reviews like these: worker A is 97% positive out of 1000 reviews and worker B is 98% positive out of 100 reviews. Then which worker is better? In term of the positive reviews, B is higher. But B gets much fewer reviews, which makes B's rating seems not as trustworthy as A's. So the number of reviews a worker gets matters. In order to predict the workers' future performance and rank them using a single metric, we designed a rating formula including both the ratings and the number of reviews using Beysian inference [7]. Beysian inference is a widely used and powerful tool. In the Beysian inference, there is a prior and a posterior. We treat how people rate a worker as a random variable P and the current set of people's ratings of a worker as a random variable X. We observe that the prior $X|P \sim B(r,p)$. Here, $B(r,p)$ is a binomial distribution [20], where r is the number of reviews and p is the probability that a worker gets a positive rating. We do not know P's distribution. But it is appropriate to assume that it follows a beta distribution $\beta(a,b)$ [17] because beta distribution covers a broad range of distributions with the variations of its two parameters a and b. So $P \sim \beta(a,b)$. Then we found that the posterior $P|X$ is still a Beta distribution but with parameters $a+x$ and $b+r-x$. That is, $P|X \sim \beta(a+x, b+r-x)$. We interpret x as the new positive ratings on the basis of the original a positive ratings and $r-x$ as the new negative ratings on the basis of the original b negative ratings. We then used the ratio of the mean and variance of the posterior Beta distribution as the review score for each worker. For the review score, the higher the value, the better the review is.

In AHP, after matrix X is ready, we do a transformation on X to normalize the data and make them contribute positively towards our preference following Step 5. After the transformation, matrix X becomes:

$$\begin{array}{c c} Choice \backslash Attribute & \begin{array}{ccc} \alpha_1 & \alpha_2 & \alpha_3 \end{array} \\ \begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array} & \begin{pmatrix} 1 & 1 & 0.67 \\ 0.33 & 0.81 & 1 \\ 0.5 & 0.92 & 0.83 \end{pmatrix} \end{array}$$

The final score matrix of the three workers $S = X\Lambda$, which is $(0.93, 0.66, 0.73)$. Sorting the workers in a non-increasing order based on their scores, the requester's preference list is: $\{w_1, w_3, w_2\}$.

VI. Worker's Preference List

In a similar way, we can generate a worker's preference list using AHP. A worker determines which tasks are most appealing to him by looking at three attributes: the reward of the task, the level of specialization required, and the time required to complete the task.

VII. Matching Tasks and Workers

Now that we have both the worker's preference list and the requester's preference list, we can match them pairwise to one another using the stable matching (SM) algorithm in Fig. 4 derived from the solution to the Stable Marriage Problem [16]. In SM, we find the best possible pairing between tasks and workers. Here 'stable' means that no worker will prefer another task than his matched one and no requester will prefer another worker than his assigned one. Algorithm SM does one-to-one matching. If the sizes of the workers and tasks are not the same, we can insert some dummies in the shorter preference list to make the two sizes equal. After running SM, if a requester/a worker gets a dummy, that means he does not get a match this time from the platform and needs to wait for future opportunities.

There are three main components in SM. In the Main control, every task t_j runs the Proposal subroutine to find its matching worker. In the Proposal subroutine, task t_j is matched with its next preference w_i. Then subroutine Refusal is called to see if w_i would refuse the match. In subroutine Refusal, if w_i is assigned t'_j but prefers t_j over t'_j, we break up w_i and t'_j and assign w_i to t_j. Now t'_j goes back to the market and needs to get a new matching worker by calling Subroutine Proposal. Otherwise, the matched pair stay matched. The algorithm terminates after all the tasks have their matched workers.

VIII. Simulations

In this section, we evaluate the quality of our proposed platform by comparing it with the existing platforms and the one that uses the random algorithm using a customized simulator written in Matlab.

Algorithm SM: Stable Matching

```
1: Inputs: a set of tasks $T$, and a set of workers $W$.
2: Output: a stable task assignment $A$ matching a worker
   with a task.
3: Each worker $i \in W$ generates his task preference list
   using AHP
4: Each requester generates his worker preference list using
   AHP
5: Main control
6: Initialize each worker as unassigned.
7: for each task $t_j \in T$ do
8:     call Subroutine Proposal for $t_j$.
9: end for
10: return $A$ as the stable task assignment.
11: Subroutine: Proposal
12: Input: task $t_j$.
13: if $w_i$ is the next entry in $t_j$'s preference order then
14:     assign $w_i$ to $t_j$.
15:     call Subroutine Refusal for $A$, $t_j$, and $w_i$.
16: end if
17: Subroutine: Refusal
18: Input: assignment $A$, task $t_j$, and worker $w_i$.
19: if $w_i$ is assigned $t'_j$ but prefers $t_j$ over $t'_j$ then
20:     break up $w_i$ and $t'_j$ and assign $w_i$ to $t_j$; update $A$.
21:     reassign $t'_j$ by calling Subroutine Proposal.
22: else
23:     the matched pair remain matched
24: end if
```

Fig. 4. The stable matching (SM) algorithm

A. Algorithms Compared

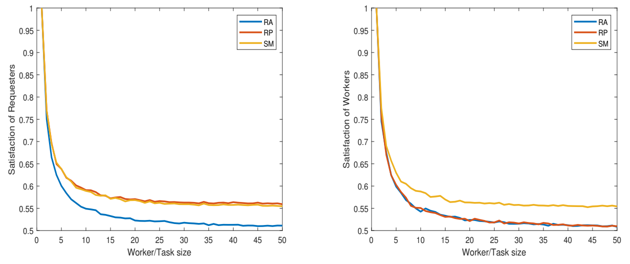
We compare the following algorithms:

- Stable Matching (SM) algorithm: it produces a stable matching between workers and tasks using the preference lists generated from AHP.
- Requester Picking (RP) algorithm: the requester handpicks his favorite worker from the platform. If the favorite worker is not available, the requester picks his next choice. This is the method used by Chegg Tutors [3] and Fiverr [5] platforms.
- Random Assignment (RA) algorithm: this is the random assignment of the tasks and the workers.

B. Comparison Metric

We assess the success of the algorithms by a satisfaction metric representing how each party i (either a worker or a task) feels about its partner after they have been matched. Let us look at i's preference list. Suppose the size of the preference list is n. The index of the partner assigned to i in the preference list is j. Then i's satisfaction score f_i is calculated as follows:

$$f_i = (n - j + 1) \cdot \frac{1}{n} \tag{2}$$



(a) Satisfaction of Task Requesters (b) Satisfaction of Workers

Fig. 5. Comparison of Stable Matching (SM), Requester Picking (RP), and Random Assignment (RA) algorithms with equal worker and task sizes

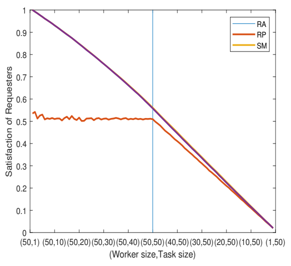
For example, suppose worker i's preference list is: $(t_3, t_5, t_1, t_4, t_2)$. The size of the preference list is 5. If the worker is assigned t_3 which has an index of 1 on the list, his satisfaction score f_i is 1. This means that he is 100% satisfied. If he is assigned the next task on the list, his satisfaction will decrease by 0.2 to 80%.

C. Simulation Setup

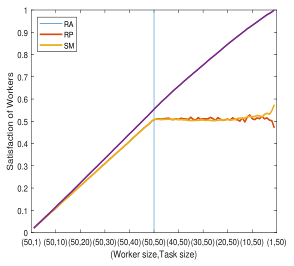
In our simulations, we have a set of tasks T and a set of workers W. For a requester, he considers three attributes to evaluate a worker: availability, previous reviews, and desired pay rate. And for a worker, he uses the reward of the task, level of specialization, and time required to finish to choose a task. In our simulation setting, when a requester adds a new task to our platform, he enters the reward of the task in the range of [\$10,\$100], level of specialization in the range of [1,5], and time required to complete the task in the range of [1,7] days. And when a worker registers with our platform, he specifies his availability in the range of [1,5] days and desired pay rate in the range of [\$10,\$20] per hour. The reviews of a worker are left by the requesters on the platform after the tasks are done. The platform calculates the review score of each worker using the Bayesian method mentioned in Section V. For each requester or worker, he also needs to fill out the PCM indicating how he weighs the pairwise importance of the attributes he uses to evaluate the other side. Since we do not have the real data from the requesters or workers, these data are randomly generated within the range of $1 - 9$ at the moment. In the simulation, we tried the number of workers/tasks from 1 to 50.

After the data are ready, we used AHP to generate the worker's preference list and the task's preference list and then feed them to the SM algorithm. The RP and RA algorithms do not need the preference lists. After the workers and the tasks are matched by the three algorithms, we calculated their satisfaction scores. In each parameter setting, we ran the three algorithms 100 times and took the average of the satisfaction scores. We have two cases in our experiments.

In our first case, there are equal numbers of tasks and workers. The simulation results are shown in Figs. 5(a) and 5(b). Fig. 5(a) compares the requester's satisfaction of the three algorithms. We can see that the requester's satisfaction is the lowest if RA is used. This is because RA does not consider



(a) Satisfaction of Task Requesters



(b) Satisfaction of Workers

Fig. 6. Comparison of Stable Matching (SM), Requester Picking (RP), and Random Assignment (RA) algorithms with unequal worker and task sizes

the requester's preference. The requester's satisfaction of SM is very close to that of RP. This means that SM has done a good job in finding a match as if the requester picked the worker himself. Fig. 6(b) shows the worker's satisfaction using the three algorithms. The workers are most satisfied in SM. The worker's satisfaction of the RP algorithm is downgraded to that of the RA because in RP, the requester does not consider the preference of the workers when he picks workers. To summarize, in order to maximize satisfaction in workers and requesters alike, SM is the best choice.

In the second case, the numbers of the tasks and workers may not be equal. Fig. 6(a) shows the requester's satisfaction when the sizes of the tasks and the workers vary. The X-coordinate represents the (worker size, task size) pair. In the middle point of the X-axis, the (worker size, task size) is (50, 50). This means there are 50 workers and 50 tasks. When we slide to the left side, the number of workers is still 50, but the number of tasks decreases by 10 at each axis tick. And when we move to the right side, the number of tasks is still 50, but the number of workers decreases by 10 at each axis tick. In the whole range of the X-axis, the requester's satisfaction scores of the SM and the RP algorithms are very close (curves overlapped) and higher than those of RA for the same reason as in the equal-size case. The requester satisfaction scores of SM and RP start from 100% when the requester has all the workers available to choose from for his task, and then decrease linearly either when the number of workers is fixed at 50 and the number of tasks increases from 1 to 50, or when the number of tasks is fixed at 50 and the number of workers decreases from 50 to 1. The requester's satisfaction falls almost to zero when there are 50 tasks and only 1 worker. In this case, the majority of the tasks will get a dummy worker, which reduces the requester's satisfaction score. As for the RA algorithm, since it does random matching, even though there are more workers than the tasks on the left part of the figure, the satisfaction scores of the data points on the left side are no different from that of the middle point. On the right side of the figure, the satisfaction scores of RA go down linearly as there are more tasks than the workers. The worker's satisfaction scores from the three algorithms are presented in Fig. 6(b) and can be interpreted in a similar way.

In summary, from our experiments we can conclude that SM can maximize the satisfaction of both the requesters and the workers by considering all their preferences.

IX. Conclusion

In this paper, we have proposed a crowdsourcing platform model that considers the satisfaction of both the requesters and the workers so as to improve the quality of the platform. We have adopted Analytic Hierarchy Process (AHP) to generate the preference lists of both sides automatically after considering multiple complex criteria deemed by each party. We have put forward the stable matching (SM) algorithm to pair the workers and the tasks so that they will not switch to other choice other than that matched. The simulation results have shown that our proposed platform can make the requesters and workers more satisfied than the existing ones and one that uses the random assignment. In the future, we will work on more ways to further improve our model.

Acknowledgments

This research was supported in part by NSF under Award #1757893.

References

[1] AHP. https://en.wikipedia.org/wiki/Analytic_hierarchy_process.

[2] Amazon mechanical turk. http://mturk.com.

[3] Chegg Tutors. https://www.chegg.com/tutors/become-a-tutor/?from_header=1.

[4] Crowdsourcing. https://en.wikipedia.org/wiki/Crowdsourcing#Crowdsourcers.

[5] Fiverr. https://en.wikipedia.org/wiki/Fiverr.

[6] J. Bentez, X. Delgado-Galvn, J. Izquierdo, and R. Prez-Garca. Achieving matrix consistency in ahp through linearization. *Applied Mathematical Modelling*, 35(9):4449 – 4457, 2011.

[7] J-M Bernardo. Reference analysis. In *Handbook of statistics*, volume 25, pages 17–90. 2005.

[8] X. Chen. A stable task assignment scheme in crowdsourcing. In *IEEE International Conference on Embedded and Ubiquitous*, pages 489–494, 2019.

[9] P. Cheng, X. Lian, L. Chen, J. Han, and J. Zhao. Task assignment on multi-skill oriented spatial crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2201–2215, 2016.

[10] D. Deng, C. Shahabi, and U. Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *21st SIGSPATIAL GIS*, pages 314–323, 2013.

[11] D. Gale and L. S. Shapley. College Admissions and the Stability of Marriage. *American Mathematical Monthly*, 69:9–14, 1962.

[12] J. P. Hanna, M. Albert, D. Chen, and P. Stone. Minimum cost matching for autonomous carsharing. In *IFAC-PapersOnLine*, volume 49, pages 254–259. 2016.

[13] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *21st SIGSPATIAL GIS*, pages 189–198, 2012.

[14] L. Kazemi, C. Shahabi, and L. Chen. GeoTruCrowd: Trustworthy Query Answering with Spatial Crowdsourcing. In *ACM SIGSPATIAL*, 2013.

[15] Y. Li, M. L. Yiu, and W. Xu. Oriented online route recommendation for spatial crowdsourcing task workers. *Advances in Spatial and Temporal Databases*, pages 137–156, 2015.

[16] D. F. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. In *Theoretical Computer Science*, volume 276, pages 261–279. 2002.

[17] K. Pearson. Mathematical contributions to the theory of evolution, XIX: Second supplement to a memoir on skew variation. *Philosophical Transactions of the Royal Society A*, 216(538-548):429–457, 1916.

[18] Thomas L. Saaty. What is the analytic hierarchy process? In *Mathematical Models for Decision Support*, volume 48, pages 109–121. Springer Berlin Heidelberg, 1988.

[19] H. To, C. Shahabi, and L. Kazemi. A server-assigned spatial crowdsourcing framework. *ACM Transactions on Spatial Algorithms and Systems*, 1(1), 2015.

[20] G. P. Wadsworth and George & Bryan. *Introduction to Probability and Random Variables*. New York: McGraw-Hill, 1960.