Software crowdsourcing task allocation method for collaborative development

ZHOU Zhuang[1], YU Dunhu[1、2] ,Liang Junjie[1]

(1.School of Computer Science & Information Engineering, Hubei University, Wuhan 430062, China;
2. Educational Informationalization Engineering Research Center of HuBei Province, Wuhan 430062, China)

Abstract: Aiming at the problem that the existing crowdsourcing software development task allocation mechanism can not effectively guarantee the inter-worker coordination while optimizing the total utility, on the basis of the bidding model, a distribution mechanism considering the three factors worker's ability, task module complexity and active time of workers is proposed. Firstly, the crowdsourcing workers are divided into multiple collaborative candidate groups based on active time. Then, the Kuhn-Munkres algorithm is used to select the optimal workers for each module from the collaborative candidate groups of each task, and the coordination candidate group replacement strategy is used to solve the allocation failure problem. Finally completing the assignment of all tasks within an allocation cycle. Experiments have shown that this mechanism has a higher allocation success rate and a total allocation effect than the sequential allocation method.

Key words: crowdsourcing; software development; collaborative candidate group; KM algorithm ;active time

1 Introduction

Crowdsourcing software development aims to make software development work no longer limited to small, isolated developer groups, but rather by multiple organizations in an organization and community[1], which enable software development work to be coordinated across time, geography, and organization, through a large and unknown group [2], Thereby reducing software development costs and improving development efficiency. Different from the traditional software development and outsourcing model, the workers in the crowdsourcing mode mainly come from the unknown network group, and the randomness of the network group makes the development ability of the crowdsourcing workers uneven. At the same time, because the personnel involved in crowdsourcing do not have a fixed cooperative relationship and cannot establish a collaborative development environment, the complete quality of the software crowdsourcing task is difficult to guarantee.

Many works of literatures indicate that there is an inevitable dependency between software crowdsourcing task modules. If the collaborative work between workers cannot be ensured, the software development process may be inefficient and software quality may be low[3-6]. Bandinclli et al. pointed out that due to the collaborative nature of software development, the success of development depends on “the quality and effectiveness of communication channels built into the development team”[7]. Other studies have shown that the risk of software project development through crowdsourcing is mainly the risk associated with teamwork, because crowdsourcing fundamentally changes the original organizational status, and project management methods and information exchange channels are not improved accordingly[8]. It can be seen that how to ensure the coordination among crowdsourcing workers has always been one of the urgent problems to be solved in the development of crowdsourcing software.

The task distribution methods of the existing commercial crowdsourcing platform are mainly divided into two categories. First, the TopCoder, Bountify and other platforms use the online competition based crowdsourcing development task allocation method which organizes the task into a competition and selects the winner (and runner-up) based on the community's peer-review of the task submission[1]. The use of a competitive mechanism can guarantee the quality of the task completion, but a competitive task with a long development cycle will cause a large loss to the loser. Therefore, this method is only suitable for micro, short-term development tasks, and it is also a difficult and time-consuming task to select winners from a large number of submitted tasks. [9]。 Second, the task allocation method based on the bidding mode adopted by GetACoder, Pig Bajie, Code City, Liberation, etc. The method realizes the two-way choice between the crowdsourcing platform and the workers and helps to achieve a stable employment relationship, which is suitable for development tasks of various sizes, but the task publisher needs to browse a large amount of bidding information and perform manual comparison selection with relatively high time and labor cost. In academia, many scholars have actively carried out research in this field. For example, Shi Zhan et al. proposed a task allocation mechanism based on user reliability[10]. Mao et al. proposed using historical data to train the classification model, and assign tasks based on the similarity between the tasks to be assigned and the static attributes of the historical completion tasks[11]. Shao et al. recommend a combination of neural networks and content-based filtering methods to recommend developers[12]. Zhu J a et al. proposed a ranking

method based on topic features to rank workers' ability to complete worker recommendation[13]. Wang Z et al. based on the capability improvement model to achieve the recommendation of crowdsourced software developers[14]. The above research is based on the premise that the crowdsourcing software development task is completely independent, emphasizing the pairing between individuals and tasks. More consideration is to decompose the tasks in the task release stage to ensure the independence of the tasks[2], but none of them further effective methods are proposed to ensure that crowdsourcing workers work together. The synergy in software crowdsourcing can be roughly divided into two categories: one is the formal coordination between workers through the exchange and correct processing of structured documents, and the other is the informal exchange of freely exchanged structured or unstructured information among workers[15]. In comparison, the second approach has lower requirements for task publishers and workers. Because of free interaction, their behavior is affected by each other which allows partners to understand each other's collaborative work[10] and give full play to the advantages of group wisdom. Therefore, it is more suitable for crowdsourcing development mode. At the spatial level, the Internet-based crowdsourcing environment has the communication infrastructure required to achieve synergy, eliminating the need for crowdsourcing workers to concentrate on the designated locations. At the time level, due to the uncertainty of the crowdsourcing community, it is difficult to ensure consistent working hours. At the same time, most people who participate in crowdsourcing are part-time and can only be active online for crowdsourcing at certain times of the day. Therefore, selecting crowdsourced workers with the same active time is the key to achieving synergy in software crowdsourcing.

In order to ensure the communication between workers and improve the efficiency and quality of crowdsourcing software development, this paper designs a crowdsourcing task allocation method suitable for multi-module software task collaborative development based on the bidding model. The task assignment is based on three factors: worker ability, task module complexity and worker active time. The crowdsourcing workers are first grouped according to the total number of active time periods required by the task, and then the workers are in the same collaborative candidate group under the same task. Then, the KM algorithm is used to optimize the global total utility, and the collaborative candidate group replacement strategy is used to solve the allocation failure problem, and finally, all tasks in an allocation cycle are allocated.

2 Software crowdsourcing task assignment

This section gives a definition of the crowdsourced software development system model and a description of the problem.

1 Definition

The crowdsourcing software development system studied in this paper is based on the bidding model, First, the task publisher needs to develop task-related information and publish the task information to the crowdsourcing platform. Developers then browse through the tasks in the crowdsourcing platform that is available for registration and select their competent and interested development tasks to sign up. After the end of an allocation period, the platform should select the tasks whose registration ends and the number of applicants meets the task requirements, then select a group of workers which meets the collaborative work requirements from each registered worker of the task to form a workgroup, Finally, select the most suitable worker for each module from the workgroup for development work. After the workgroup completes the development work, the module code is submitted to the platform, and the platform integrates all the codes to form a complete solution. The relevant concept definitions for the system model are given below:

Def 1 (Crowdsourcing software development task). The crowdsourcing software development task studied in this paper refers to the code writing task that has been divided into several smaller modules which described by UML block diagram. It is described as a four element set: , where is the modules set of task; is the task type; is a time threshold that indicates how long the task requires workers to work together every day; is a collection of workers who successfully registered for the task.

Def 2 (Task module). The task module refers to the development module obtained by dividing the crowdsourcing software development task requirements based on functions, and is described as a two element set:, where is a module description, including module requirements, rewards and other information; is the expected complexity of the module.

Def 3 (Crowdsourcing worker). A crowdsourcing worker is a participant in a crowdsourcing software development task, described as a two element set:,where is the crowdsourcing worker's development ability set for different types of tasks, indicates the development capability evaluation value of the platform for the crowdsourcing worker to complete the category task; is the active time set that crowdsourcing workers are more inclined to carry out development work during these periods, , represents the hour of the 24 hours a day.

Def 4(Distribution utility). The distribution utility is a measure of the distribution effect. It is generally believed that this will generate higher value when assigning more capable workers to more difficult modules. The allocation utility calculation method of the module *m* assigned to the worker is: .

2 Description of the problem

The problem to be solved in this paper is to design a task assignment method that optimizes the pairing utility to optimize the target while achieving the optimal matching of the developer-task module under multitasking.

Assume that when the platform performs task assignment, the set of tasks that satisfy the condition is , The final assignment result of task is ,where indicates that module was assigned to developer . First of all, in order to meet the collaborative development requirements between task modules, the number of active sessions that developers need to have is not less than . Suppose , Common active time period is expressed as set， it must satisfy:

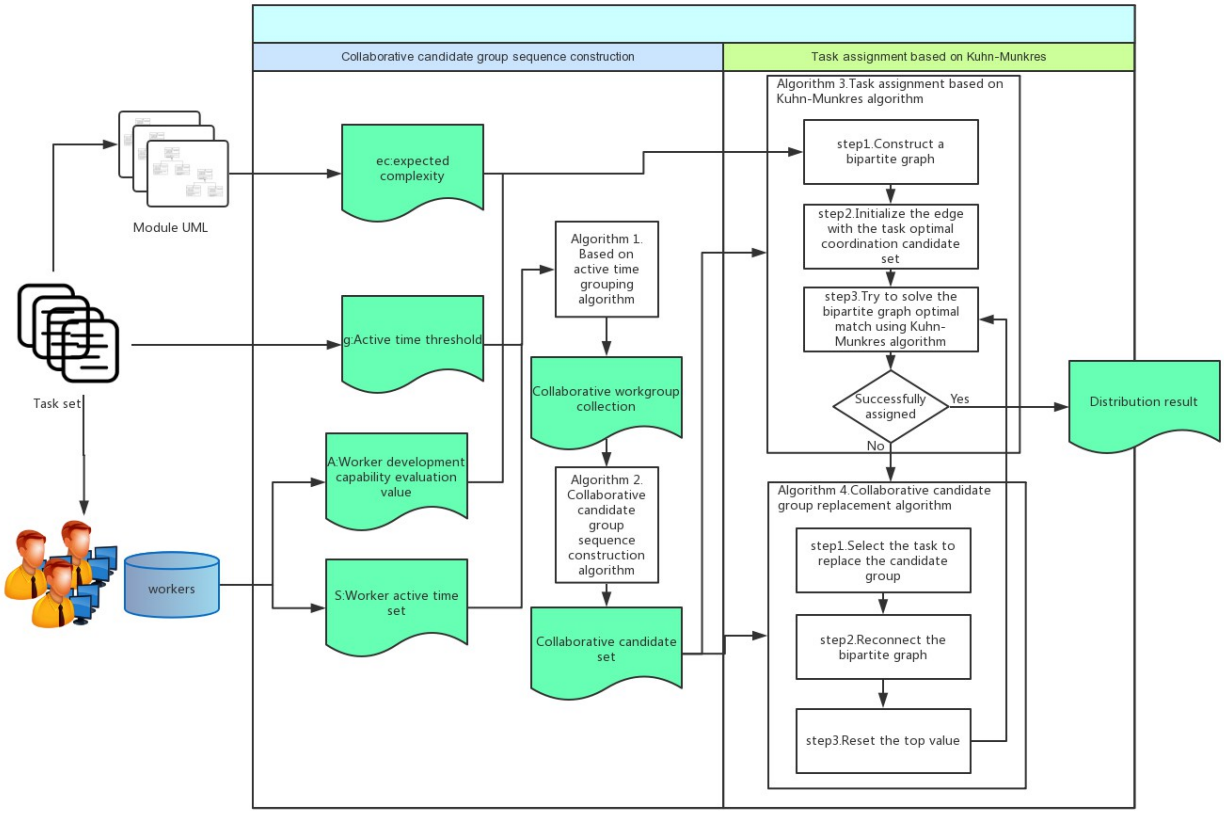
The total utility of the assignment result of a single task t is:

The goal of this paper is to maximize the sum of the total utility of all tasks in a distribution cycle, which can be formalized as the following optimization problem:

The first constraint ensures that the final selected developer has already signed up for the task. The second constraint ensures that the number of active sessions shared by the developer is not less than , which ensures that workers can collaborate in the same workgroup. It is referred to as collaborative constraints. The third constraint guarantees that any developer can only be selected by one task in the result.

3 Task assignment process

This section gives the specific flow of the crowdsourcing software task assignment method proposed in this paper. The input to the allocation process is a set of assignable tasks that contain all the tasks required to meet the number of enrollees in the current allocation cycle. The output of the allocation process is the final assignment result, which is a collection of module-worker pairs. The detailed allocation process is shown in Figure 2.



Pic 2 Crowdsourcing software development worker distribution mechanism process

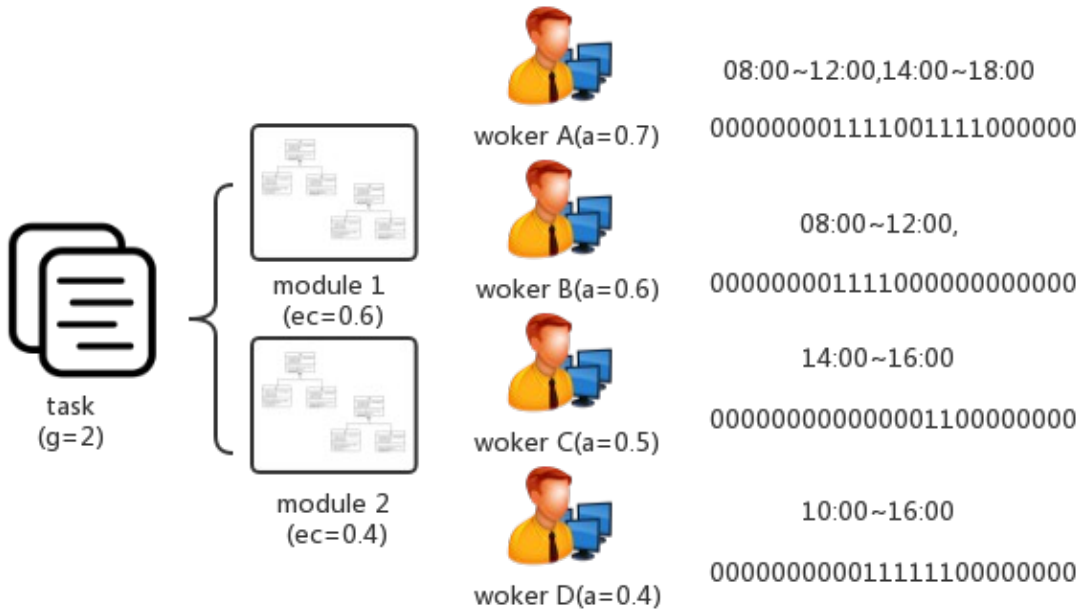
The allocation process can be divided into two steps:

(1) Constructing a collaborative candidate group sequence: In this part, each task in the input will be separately constructed as a collaborative candidate group, and the registered worker set of each task will be divided into a plurality of subsets satisfying the cooperative constraint, and finally the subset will be constructed as a specific priority sequence for subsequent matching.

(2) Optimal matching based on the Kuhn-Munkres algorithm: In this part, a bipartite graph model will be constructed for all task modules and workers. After initializing the joints of the bipartite graph, the Kuhn-Munkres algorithm is continuously tried to solve the optimal matching scheme. When the matching fails, Attempts to perform a collaborative candidate group replacement policy and rematch until the match is successful or fails because the replacement policy can no longer be executed.

4 Collaborative candidate group sequence construction

In order to ensure that a group of crowdsourced workers assigned to the same task meet the specified simultaneous online requirements, it is necessary to ensure that the crowd-workers have the same set of active time slots as possible. Therefore, we limit the total active time period threshold g, and the required crowdsourcing workers need to have a total active time period greater than or equal to g. As shown in Figure 3, a task consists of two modules (requires two workers to develop) with a time threshold g = 2, which requires the assigned worker to have a total active time of at least two hours. Then, from the situation of the four workers in the picture: A and B have a total of 4 hours from 08:00 to 12:00, and any two of them can meet the requirements; A and C have a total of 14:00~16:00. 2 hours, meet the requirements; A and D have a total of 2 hours from 09:00 to 12:00 and 2 hours from 14:00 to 16:00, and one or two hours from each of the two paragraphs can also meet the requirements; B and C have no common time period and do not meet the requirements.; B and D have a total of 2 hours from 10:00 to 12:00, which meets the requirements; C and D have 2 hours from 14:00 to 16:00, which meets the requirements. Therefore, the workers who meet the synergistic requirements have 5 kinds of {A, B}, {A, C}, {A, D}, {B, D}, {C, D}.



Pic 3 Task and worker example

When the number of workers is large, we can't exhaust all the combinations that satisfy the conditions. Therefore, we use the grouping algorithm to divide the worker set into multiple collaborative workgroups according to whether the active time contains a specific time period. The number of divisions is fixed, regardless of the number of workers. First, the crowdsourcing workers are sorted in descending order according to the ability evaluation value. Then, 24 sets of workers representing different time periods are generated, each set containing all workers whose active time includes the time period they represent. Then, all the possible combinations of the time periods are selected in the 24 time periods. At each iteration, the intersection of the corresponding worker sets in each time period of the time period combination of the iteration is calculated. If the number of workers in the intersection is greater than or equal to the number of the task modules, the intersection is a qualified collaborative workgroup.

The specific algorithm for calculating the collaborative workgroup is shown in Table 1:

Table 1 Active time based grouping algorithm

Algorithm 1. Active time based grouping algorithm
input: Registered developer collection， Threshold， Number of task modules
output: Collaborative workgroup collection
1. Sort developers by ability of workers in the collection by descendant order
2. Generate 24 worker sets
3. for each do
4. for each do
5.
6. end for
7. end for
8. Generate all combinations of time periods from 24 time periods as
9.
10. for each do
11.
12. for each do
13.
14. end for

15. if then
16.
17. end for
18. return

Anyone of the collaborative workgroups in meets the requirements for distribution. As shown in FIG. 3, the workers in FIG. 2 are grouped to obtain four cooperative workgroups.

Table 2 Collaborative workgroup example

Collaborative time period (g=2)	Collaborative workgroup
{9,10},{9,11},{9,12},{10,11}, {10,12}	{A,B}
{11,12}	{A,B,D}
{11,15},{11,16},{12,15},{12,16}	{A,D}
{15,16}	{A,C,D}

Obviously, the result of the assignment of task t must be one of the above-mentioned collaborative working groups or a subset thereof, but when the number of workers is large, the number of these subsets is also very large. In fact, Any worker set containing the optimal solution is equivalent to the allocation algorithm. Because the allocation algorithm will do its best to examine the workers in the group to resolve conflicts and ensure maximum allocation utility. For example, if the optimal allocation of the task t assignment algorithm is {A, D}, then the result of assigning {A, B, D} or {A, D} or {A, C, D} to the task t is identical. However, we cannot predict the distribution results of the best utility or the collaborative workgroup in which it is located. But the total utility of the distribution is limited by the ability of the workers in the group. We can limit the utility's lower bound (the worst allocation effect that the group can assign to the task) of the assigned workers, thereby limiting the total utility 's lower bound of the allocation. For a group of workers G, this lower bound is recorded as . In the allocation, the lower bound is gradually reduced until the distribution solution can be obtained, thereby ensuring that the total allocation utility is superior. The lower bound of the distribution utility is determined by the worker with the lowest ability in the group, and the greater the number of workers in the group when the lower bound is determined, the stronger the ability to resolve conflicts. Therefore, we can start from the most capable workers in the group, gradually join the workers with the second highest ability, and further refine the collaborative workgroup into a collaborative candidate group with a smaller gap between the groups. All the collaborative candidate groups are then generated as a candidate group sequence order by . When there are more workers, the number of collaborative candidate groups may be more, and the candidate group workers with lower ranking may have a lower ability, and the allocation process is likely to end when using the earlier candidate group. Therefore, this paper sets the threshold number k of the collaborative candidate group, which can speed up the construction efficiency of the collaborative candidate group sequence.

For example, the collaborative workgroup in Table 2 can be subdivided into the corresponding collaborative candidate groups in Table 3. Assuming that the worker's ability ranking is A>B>C>D, then the generated collaborative candidate group sequence is: ({A,B},{A,C},{A,D},{A,B,D},{A,C,D}).

Table 3 Collaborative candidate group partitioning example

Collaborative workgroup	Collaborative candidate group
{A,B}	{A,B}
{A,B,D}	{A,B },{A,B,D}
{A,D}	{A,D}
{A,C,D}	{A,C },{A,C,D}

A collaborative candidate group sequence construction algorithm for generating a collaborative candidate sequence from a collaborative workgroup collection is proposed here. First, initialize a min-heap of size . Then obtaining collaborative candidate groups by intercepting a subset of sequence numbers for each in, Calculate the worst allocation effect of each candidate group , execute the following branches in conditioned order:

1） This candidate group has been examined, skipped directly

2） The number of elements is less than k, press into

3） The number of elements is enough, if is greater than the worst allocation utility of the top candidate group, the top group will be popped and the will be pressed into the .

The specific algorithm is as follows:

Table 2 Collaborative candidate group sequence construction algorithm

algorithm 2. Collaborative candidate group sequence construction algorithm
input: Collaborative workgroup collection , Sequence size threshold
output: Collaborative candidate sequence
1. Initialize the min-heap with a capacity of
2. for each do
3. for to do
4. Group workers with subscripts from 1 to in form collaborative candidate group
5. if has not been investigated then
6. if the size of is less than k then
7. push into
8. continue
9.
10. if then
11. push into
12. else
13. break
14. end for
15. end for
16. while is not empty
17. add the top of to the collaborative candidate sequence
18. return

5 Task assignment based on Kuhn-Munkres algorithm

The crowdsourcing software task assignment problem can be refined into the matching problem between the module and the crowdsourcing workers. Simply considering the goal of optimizing the total utility, the problem can be modeled as a match the problem of a weighted bipartite graph with the task module-worker allocation utility as the weight. The classical algorithm for solving this problem is the KM (Kuhn-Munkres) algorithm[16]. The allocation algorithm of this paper is based on KM algorithm. KM algorithm is used to solve the optimal matching when the bipartite graph structure is determined, and the optimal matching of the cooperative constraint is obtained by dynamically changing the bipartite graph structure through the collaborative candidate group replacement strategy.

The algorithm can be roughly divided into two main parts:

a) Construct and initialize a bipartite graph model, then try to solve the bipartite graph optimal matching using the KM algorithm.

b) Perform a cooperative candidate group replacement strategy when the midway matching fails, rematch until the allocation is successful. The allocation fails after all the candidate candidates have been exhausted.

4.1 Construct a bipartite graph for optimal matching

After the group of registered workers of all tasks in one allocation period is grouped based on active time, Set the left vertex set to the union of the module sets of all tasks, Set the right vertex set to the union of the optimal collaborative candidate groups for all tasks, Set the right and left set points and the edge weights to the assign utility . Join all the edge with non-zero weights and complete the bipartite graph construction. Then try to solve the bipartite graph optimal match using the KM algorithm. The process is outlined below:

a) Set the left vertex set to the union of the module collections for all tasks.

b) Set the right vertex set as the union of the optimal collaborative candidate groups for all tasks.

c) Set the left and right set vertex edge weights to assign utility, connect the edges whose weights is not zero.

d) Try to use KM algorithm to solve the bipartite graph optimal matching. If it succeeds, output the matching result. If it fails, replace the coordinating candidate group of a task by the replacement strategy, and re-match until the matching success or there is no longer coordinating candidate group for replacing.

The specific steps of the matching algorithm are as follows:

Table 5 KM-based task assignment algorithm

algorithm 1. Task assignment algorithm based on Kuhn-Munkres
input: task set T、Collaborative candidate sequence set
output: distribution result
1. initialize the min-heap with a capacity of k

```
2.
3.  for each  do
4.
5.      for each  do
6.
7.          for each  do
8.
9.              connect the edges corresponding to m and d, and set the weight to
10.
11.         end for
12.     end for
13. end for
14. using KM algorithm to solve the optimal match of bipartite graph of
15. if Match failed then
16.     Performing a collaborative candidate group replacement strategy
17.     If Successful replacement then
18.         go back to step 16 to re-allocate
19.     else
20.         assignment failed, end algorithm
21. return distribution result
```

4.2 Collaborative candidate group replacement strategy

The reason why the KM algorithm failed is that the search for the augmenting path of a module vertex fails and the equal subgraph cannot be expanded. The mechanism of this paper has built a sequence of collaborative candidate groups for each task before the allocation. The algorithm allows changing the structure of the bipartite graph by replacing the collaborative candidate group based on the sequence. This makes it possible to retry the conflict that the original bipartite graph could not handle. All collaborative candidate groups have met the collaborative time constraint, and the replacement strategy will use greedy thinking to find the replacement of the collaborative candidate group that is expected to minimize the loss of the total allocation utility. The process is outlined below:

a) Calculate the worst-cased utility difference between the currently used collaborative candidate group and the sub-optimal collaborative candidate group for each task involved, and select the task with the smallest difference.

b) Clear all edges of modules in taskhave been connected.

c) Set the top label of each module in task to , where is the optimal worker of the current collaborative workgroup of .

The detailed algorithm is as follows:

Table 6 Collaborative candidate group replacement algorithm

```
algorithm 1. Collaborative candidate group replacement algorithm
input: Involved task set , Collaborative candidate sequence set ,current bipartite graph
output: Replacement result
1.
2.
3.  for each  do
1.
2.      if then
3.
4.  end for
5.  if  then
6.
7.      for each  do
```

```
8.              delete all existing  related edges in
9.              for each  do
10.                 connect the edges corresponding to  and , and set the weight to
11.             end for
12.         end for
13.   else
14.         replacement failed
```

6 Experiment and result analysis

In order to verify the effectiveness of the algorithm, we captured the information of 28,834 crowdsourced workers on the programmer's inn as experimental data sets. The worker information includes the worker's work time and skill level scores. In the experiment, the task data is generated by computer simulation. The number of modules included in each task is subject to the distribution, and the complexity of the module is subject to the distribution. The worker data uses the actual data set obtained. It is important to note that we normalize the worker rating level as the worker development capability value. This experiment runs on a machine with a 2.4GHz Inter(R) Core(TM)i5 processor and 8GB of RAM. The operating system is Windows 10 and the programming language is Java.

The object of the experiment comparison is the sequential allocation method, which assigns each task in order (in random order in the experiment), selects the worker who meets the time constraint and assigns the best utility to match the task module, and eliminates it in the subsequent task assignment. Workers were assigned to avoid workers being assigned multiple times.

5.1 Total utility comparison

This experiment compares the proposed method and the single task priority allocation method （sequential allocation method） in the total utility of the distribution. Table 5-7 shows the results of the three sets of experiments and the parameter settings.

In Table 5-7, the first four columns represent the four experimental parameters of the number of tasks, the number of modules, the threshold number of shared active periods g, and the number of workers. The number of modules is related to the number of tasks, and is only for reference. The fifth and sixth columns are the total utility of the proposed method and the single task priority allocation method. The last column reflects the improvement ratio of the relative order allocation method of the algorithm in the total utility.

These three tables show the impact of changes in the number of workers, the number of tasks, and the threshold g on the total utility of the allocation. In order to make the experimental results more general, each experiment was repeated 10 times (only the distribution was successful), and the workers were randomly selected from the data set each time. The average of the total utility assigned will be shown in the table.

From the results of Table 5-7, in the case of various parameters, the proposed algorithm always outperforms the sequential allocation method in optimizing the total utility of allocation. And as the parameters change, there are the following conclusions:

1) As other conditions remain unchanged, when the number of workers participating in the distribution is small, the improvement ratio of the proposed algorithm is relatively high. In Table 5, we can observe up to 25% improvement. It shows that when the number of workers is small, the “waste” phenomenon that may occur in the sequential allocation (the more powerful workers are assigned to the modules with lower complexity due to the order of distribution) will have a greater impact on the total utility. When the number of workers increases, the redundant higher level workers increase, and even if there is a waste, there are fewer redundant workers to choose from, so the impact on the total utility is small. The method in this paper uses a KM algorithm that accurately solves the optimal allocation of a particular bipartite graph, which can be relatively stable and maintain a relatively high total utility.

2) In general, when other conditions remain unchanged, the more tasks are assigned, the higher the improvement ratio of the proposed algorithm. It is because the task assignment result that is first assigned in the sequential allocation method will affect the subsequent task assignment, and as the number of tasks increases, this effect will gradually overlap. At the same time, since the sequential allocation only focuses on the optimization of the single task, the impact on the subsequent allocation is generally negative, so the negative impact on the total utility is greater after multiple times of superposition.

2) When other conditions remain unchanged, the higher the active time threshold g, the higher the improvement ratio of the proposed algorithm. It can be seen from Table 7 that when the threshold g is increased, the

total utility of both methods is reduced because when the synergistic conditions become harsh, it is more difficult to ensure that the workgroup that satisfies the coordinated constraint is at a higher level in the worker group. For the sequential allocation, the effect of "waste" is more serious in the case of fewer workers satisfying the synergistic constraint, resulting in a lower overall utility of the final allocation.

Table 7 Change the number of workers

	Task	Modules	g	Worker	proposed algorithm		Sequential allocation	rate
					execution time(s)	total utility	total utility	
ex1-1	20	117	3	150	6.37	29.3520472	23.3821231	25.53%
ex1-2	20	117	3	200	7.45	31.3905924	26.7513805	17.34%
ex1-3	20	117	3	250	9.56	33.0163823	29.0693901	13.58%
ex1-4	20	117	3	300	12.28	33.7066428	30.5247659	10.42%
ex1-5	20	117	3	350	14.79	34.7102485	31.6481798	9.68%
ex1-6	20	117	3	400	18.99	34.89492	32.0724621	8.80%
ex1-7	20	117	3	450	24.55	36.0265632	33.4805254	7.60%
ex1-8	20	117	3	500	29.27	36.3851108	33.9265583	7.25%
ex1-9	20	117	3	550	38.65	36.6911361	34.2737944	7.05%
ex1-10	20	117	3	600	43.36	37.3118961	35.0473219	6.46%
average	20	117	3	375	20.52	34.34855394	31.01765015	11.37%

Table 8 Change the number of tasks

	task	modules	g	worker	proposed algorithm		Sequential allocation	rate
					execution time (s)	total utility	total utility	
ex1-1	10	56	3	500	17.98	18.7633905	18.3278971	2.38%
ex1-2	15	94	3	500	23.81	31.3564197	30.3893707	3.18%
ex1-3	20	138	3	500	37.65	51.7374994	50.0257427	3.42%
ex1-4	25	144	3	500	50.82	51.3522836	49.4163188	3.92%
ex1-5	30	160	3	500	69.63	57.9694126	55.5075413	4.44%
ex1-6	35	199	3	500	142.30	59.6579574	57.4535496	3.84%
ex1-7	40	258	3	500	292.59	75.6242279	71.3790431	5.95%
ex1-8	45	263	3	500	292.12	91.251471	85.0746393	7.26%
ex1-9	50	308	3	500	552.74	90.9060153	83.2345715	9.22%
ex1-10	55	334	3	500	819.42	90.5350776	85.2181511	6.24%
average	33	195	3	500	229.91	61.9153755	58.60268252	4.98%

Table 9 Change the active time threshold g

	task	modules	g	worker	proposed algorithm		Sequential allocation	rate
					execution time(s)	total utility	total utility	
ex1-1	20	140	1	200	14.93	37.830542	35.1537774	7.61%
ex1-2	20	140	2	200	22.01	37.5876634	34.9651329	7.50%
ex1-3	20	140	3	200	26.11	36.6100334	34.2180012	6.99%
ex1-4	20	140	4	200	28.29	35.5975418	32.6545606	9.01%
ex1-5	20	140	5	200	38.56	33.6717179	30.0582529	12.02%
ex1-6	20	140	6	200	46.96	33.3839609	29.7799326	12.10%
average	20	140	4	200	29.47	35.78024323	32.80494293	9.21%

5.2 Distribution success rate comparison

This experiment compares the distribution success rate between the proposed method and the single task priority allocation method (sequential allocation method ）. Table 8-10 shows the results of the three sets of experiments and the parameter settings.

In Table 8-10, the meaning of the first four columns of parameters is the same as in section 5.1. The fifth and sixth columns are the distribution success rates of the proposed algorithm and the sequential allocation method.

These three tables show the impact of changes in the number of workers, the number of tasks, and the

threshold g on the success rate of distribution. Each experiment is performed 100 times and the table shows the number of times the assignment was successful. It is important to note that in order to ensure the existence of a feasible solution for each experiment assignment, the worker data for the three sets of experiments in this section is generated in two parts:

1) Generate as many workers as the number of modules for each task: first, select a continuous period of length of the task common active time threshold g as the active time of all workers, and then add randomly consecutive periods (0-8) to each worker's active time. Generated workers' ability value obey . This part of workers guarantees that there must be a solution to the task assignment.

2) Workers randomly selected from the real worker dataset: randomly select a specific number of workers from the real data set according to the experimental needs. This part of the workers makes the workers involved in the distribution redundant.

From the results of Table 8-10, the advantages of the proposed algorithm are very obvious in the allocation success rate. At the same time, as the parameters change, the following conclusions can be drawn:

1) As the number of workers increases, the success rate of distribution of both methods increases. It can be seen from Table 8 that when the number of workers is low (only a little more than the number of modules), the distribution success rate of the two groups is very low, the success rate of the sequential allocation method is almost zero, and the proposed algorithm is only 16 %, which means that neither method can find the optimal solution. However, when the number of workers is gradually increased, the method can achieve a success rate of 98% when the number of redundant workers reaches 15 or so. When the number of redundant workers is higher than 25, the success rate reaches 100%. But the sequential allocation requires more redundant workers to achieve a higher allocation success rate because it fails to consider the impact of individual assignments on other task assignments.

2) When the number of tasks increases, the success rate of the proposed algorithm will gradually increase. This is because if the total number of modules is unchanged and the number of tasks increases, the equivalent of disguised reduces the number of modules in a single task, which leads to the coordination requirements of individual tasks lower, so that more combinations can be utilized, and thus the distribution success rate increases. However, the sequential allocation algorithm does not have obvious rules for assigning success rate changes when the number of tasks increases. In fact, it behaves more like too relevant to random data, so it is also very random in the distribution success rate.

3) When the threshold g is increased, the distribution success rate of both will decrease. For the proposed algorithm, the decrease is slower and can maintain a higher success rate when the threshold is lower. The sequential allocation method will decrease the success rate with the increase of the threshold, and the overall success rate is lower. This is because the main reason for the allocation failure is that the cooperative constraint defined by the threshold g cannot be satisfied, and the increase of the threshold g makes the constraint condition more difficult to satisfy, and thus the allocation success rate decreases. Since the method can save by modifying the allocated scheme when the single task assignment fails, and the sequential allocation method has no corresponding measures, so the sequential allocation method is more affected when the threshold g is increased.

Table 10 Change the number of workers

	task	modules	g	worker	proposed algorithm	Sequential allocation
					Success rate (%)	Success rate (%)
ex1-1	20	117	3	122	16	0
ex1-2	20	117	3	127	66	0
ex1-3	20	117	3	132	98	0
ex1-4	20	117	3	137	98	0
ex1-5	20	117	3	142	100	1
ex1-6	20	117	3	147	100	8
ex1-7	20	117	3	152	100	24
ex1-8	20	117	3	157	100	64
ex1-9	20	117	3	162	100	82
ex1-10	20	117	3	167	100	89
average	20	117	3	145	87.8	26.8

Table 11 Change the number of tasks

	task	modules	g	worker	proposed algorithm	Sequential allocation

					Success rate (%)	Success rate (%)
ex1-1	10	100	3	110	26	3
ex1-2	15	100	3	110	69	0
ex1-3	20	100	3	110	83	15
ex1-4	25	100	3	110	89	42
ex1-5	30	100	3	110	100	0
ex1-6	35	100	3	110	100	0
ex1-7	40	100	3	110	100	0
ex1-8	45	100	3	110	100	0
ex1-9	50	100	3	110	100	100
ex1-10	55	100	3	110	100	0
average	33	147	3	110	86.7	16

Table 11 Change the active time threshold g

	task	modules	g	worker	proposed algorithm	Sequential allocation
					Success rate (%)	Success rate (%)
ex1-1	10	55	1	64	97	69
ex1-2	10	55	2	64	81	29
ex1-3	10	55	3	64	66	20
ex1-4	10	55	4	64	39	8
ex1-5	10	55	5	64	24	8
ex1-6	10	55	6	64	14	3
average	10	55	3	64	53.5	23.17

7 Conclusion

This paper models the collaborative software task assignment problem in the crowdsourcing environment as the distribution optimization problem, and integrates the three factors of worker capacity, task module complexity and worker active time to establish optimization goals. And based on the Kuhn-Munkres algorithm, the optimization problem is solved by introducing a collaborative workgroup replacement strategy. The test results show that the proposed method can increase the total utility by about 25% and the average success rate by about 30% compared with the sequential allocation method.

Crowdsourcing is the product of making full use of the wisdom of the Internet community. Due to the randomness of the developer community on the Internet and the complexity of the software development process, it is difficult to determine the pros and cons of task assignment. In fact, the utility of the algorithm is based on the accuracy of the worker and task matching metrics, and the proposed algorithm simplifies the matching utility, considering only the worker ability. Subsequent research needs to combine more factors, such as the character of the worker, the way of working, etc. into the consideration of the distribution utility, and improve the evaluation model of the distribution effect.

References

[1] Begel A, Bosch J, and Storey M A. Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder[J]. IEEE Software, 2013, 30(1):52-66. doi: 10.1109/MS.2013.13.

[2] Stol K J and Fitzgerald B. Two's company, three's a crowd: a case study of crowdsourcing software development[C]. Proceedings of the 36th International Conference on Software Engineering. ACM, 2014:187-198.doi: 10.1145/2568225.2568249.

[3] STOL K J, FITZGERALD B. Researching crowdsourcing software development: perspectives and concerns [C]// Proceedings of the 2014 International Workshop on Crowdsourcing in Software Engineering. New York:ACM. 2014:7-10.

[4] PENG X, BABAR M A, EBERT C. Collaborative software development platforms for crowdsourcing [J]. IEEE Software, 2014, 31(2):30-36.

[5] DWARAKANATH A, CHINTALA U, SHRIKANTH N C, et al. Crowd build: a methodology for enterprise software development using crowdsourcing[C]//Proceedings of the 2015 IEEE/ACM, International Workshop on Crowdsourcing in Software Engineering. Piscataway，NJ:IEEE, 2015:8-14.

[6] TAJEDIN H,NEVO D. Determinants of success in crowdsourcing software development [C]// Proceedings of the 2013 Conference on

Computers and People Research. New York: ACM. 2013:173-178.

[7] BANDINELLI S, NITTO E D, FUGGETTA A. Supporting cooperation in the SPADE-1 environment [J]. IEEE Transactions on Software Engineering, 2002, 22(12):841-865.

[8] LIANG L,TANG Y. Overview on collaborative software engineering[J]. Computer Integrated Manufacturing Systems, 2003, 9(s1):1-5.

[9] FU Y., CHEN H., SONG F. (2015) STWM: A Solution to Self-adaptive Task-Worker Matching in Software Crowdsourcing. In: Wang G., Zomaya A., Martinez G., Li K. (eds) Algorithms and Architectures for Parallel Processing. ICA3PP 2015. Lecture Notes in Computer Science, vol 9528. Springer, Cham.DOI: 10.1007/978-3-319-27119-4_27.

[10] SHI Z, XIN Y, SUN Y E, et al. Task allocation mechanism for crowdsourcing system based on reliability of users[J]. Journal of Computer Applications, 2017, 37(9): 2449-2453.

[11] MAO K, YANG Y, WANG Q, et al. Developer recommendation for crowdsourced software development tasks [C]// Service-Oriented System Engineering. IEEE, 2015:347-356.

[12] SHAO W, WANG X, JIAO W. A developer recommendation framework in software crowdsourcing development[M]. Software Engineering and Methodology for Emerging Domains. Springer Singapore, 2016.DOI: 10.1007/978-981-10-3482-4_11.

[13] ZHU J, SHEN B, HU F. A learning to rank framework for developer recommendation in software crowdsourcing[C]// Proceedings of the 2016 Software Engineering Conference. Piscataway，NJ: IEEE, 2016:285-292.

[14] WANG Z, SUN H, FU Y, et al. Recommending crowdsourced software developers in consideration of skill improvement [C]// Proceedings of the 2017 IEEE/ACM International Conference on Automated Software Engineering. Washington, DC: IEEE Computer Society, 2017:717-722.

[15] Bischofberger W R, Kofler T, Matzel K U, et al. Computer supported cooperative software engineering with Beyond-Sniff[C]. Software Engineering Environments. IEEE, 1995:135-143.

[16] Kuhn H W. The Hungarian method for the assignment problem[J]. Naval Research Logistics, 2010, 2(1-2):83-97.