Received February 28, 2019, accepted March 7, 2019, date of publication March 14, 2019, date of current version April 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2905054

Crowdsourcing Software Task Assignment Method for Collaborative Development

DUNHUI YU[1,2], ZHUANG ZHOU[1], AND YI WANG[1]
[1]School of Computer Science and Information Engineering, Hubei University, Wuhan 430062, China
[2]Hubei Engineering Research Center of Educational Informationalization, Wuhan 430062, China

Corresponding author: Zhuang Zhou (zhouzhuang1994@foxmail.com)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1003801, in part by the National Natural Science Foundation of China under Grant 61572371 and Grant 61702377, and in part by the Science and Technology Innovation Program of Hubei Province under Grant 2018ACA13.

ABSTRACT Software crowdsourcing is an emerging and promising software development model. It is based on the characteristics of Internet community intelligence, which makes it have certain advantages in development cost and product quality. Companies are increasingly using crowdsourcing to accomplish specific software development tasks. However, this development model still faces many challenges. One of the key issues is the collaboration between crowdsourced workers. Developer collaboration is important to software development, but workers in crowdsourcing come from an undefined network community, so it's hard to guarantee that they can work together. This paper focuses on task assignment and uses the active time of workers as the basis of grouping to provide a solution for multi-task to multi-worker allocation. Based on the on-demand distribution model, this paper considers three factors: worker's ability, task module complexity, and worker's active time. First, the workers are divided into multiple collaborative candidate groups based on active time. Then, the Hungarian 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 assignment failure problem. Finally completing the assignment of all tasks within an assignment cycle. The experiments have shown that the proposed method increases the total utility by 25% and the success rate of distribution by 30% than the sequential assignment method. The proposed method can give a reasonable solution for software crowdsourcing task allocation based on collaborative development.

INDEX TERMS Crowdsourcing, software development, collaborative candidate group, Hungarian algorithm, active time.

I. INTRODUCTION

Crowdsourcing software development aims to make software development no longer limited to a small number of isolated developers, but work in collaboration with developers from various organizations and communities [1]. This pattern enables software development tasks to be accomplished through collaboration of developer groups, on the premise of overcoming time, regional, and organization constraints, thus reducing development costs and efficiency [2]. Different from the traditional software development and outsourcing mode, workers in the crowdsourced mode mainly come from the groups on the network. The randomness of the network developers makes the development ability of workers uneven. At the same time, the quality of tasks cannot be ensured, because of the unfixed cooperative relationship between works and the lack of collaborative development environment [3]–[4].

Many literatures have pointed out that there is inevitable dependency between tasks. Failure to ensure the collaborative work among workers may lead to inefficient and poor software development [5]–[11]. Given the collaborative nature of software development, Bandinclli et al. suggested that the success of development depends on "the quality and effectiveness of communication between members of the development team" [12]. Although crowdsourcing development has fundamentally changed the original organizational status, project management methods and information

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad Anwar Hossain.

communication channels have not been improved accordingly. Previous studies have also shown that the risk of crowdsourcing software development is mainly related to the risk of teamwork. Therefore, how to ensure the coordination between workers has always been one of the urgent problems to be solved in the crowdsourcing software development.

The task assignment methods of existing commercial crowdsourcing platforms are mainly divided into two categories. The first one is based on online competition, organized by the contest, and selects the winner (and runner-up) based on the community peer-review of the task submission [1]. Such as TopCoder [13], Bountify. This method can guarantee the quality of the tasks, but the competitive tasks with a long development cycle will cause great loss to the losing candidates. Therefore, this method is only suitable for micro, short-term development tasks, and it is also difficult and time-consuming to select winners from a large number of submitted tasks [14]. The other is based on the bidding mode, such as GetACoder [15], zbj.com, codemart, jointForce, etc. The method realizes the two-way choice between crowdsourcing platform and workers, which is helpful to achieve stable employment relationship. It is suitable for development tasks of various scales. However, the task publishers need to browse a large amount of bidding information and perform manual comparison. The time and labor cost of this method is relatively high [16].

In academia, many scholars have actively carried out research in this field. For example, Shi *et al.* [17] proposed a task assignment mechanism based on user reliability. This mechanism takes into account the differences in worker reliability, but ignores the differences in task difficulty. Mao *et al.* [18] proposed using historical data to train classification model, and assigned new tasks based on the similarity between the tasks to be assigned and the historical tasks. This method relies on the validity of historical data, and it ignores the collaboration of worker. Shao *et al.* [19] attempted to recommend developer by combining neural networks and content-based filtering approach. It relies on intrinsic properties, but not taking into account developers' dynamic development capabilities Zhu *et al.* [20] proposed a ranking method based on topic features to rank workers' ability, and then recommended workers. But it also ignores the collaboration of worker. Wang *et al.* [21] achieved the recommendation of crowdsourcing software developers with the capability improvement model. It is also limited to the situation where a single worker completes a single task. Machado *et al.* [22] formalize task allocation in crowdsourcing scenarios as an artificial intelligence planning problem. But it does not apply to collaborative environments. Lee *et al.* [23] propose a task distribution method try to minimize the difference between the level of skill of workers and the difficulty of the tasks. But the allocation is defined as one-to-one. Roy *et al.* [24] integrate multiple human factors, such as worker expertise, minimum wage requirements and acceptance ratio, into the assignment process. But it assumed a wiki-like collaboration model e in which workers collaborate on the task itself, but not directly interacting with one another. Boutsis and Kalogeraki [25] develop a crowdsourcing system called CRITICAl and solve a task assignment problem, that efficiently determines the most appropriate group of workers to assign for each incoming task. But its constraint is based on application real-time demands. Rahman *et al.* [26] adapt various individual and group based human factors critical to complex collaborative tasks, and propose a set of optimization objectives. It realized the problem of worker collaboration, but judge it by affinity among workers.

Above mentioned work is based on the premise that the crowdsourcing software development task is completely independent, emphasizing the pairing between individuals and tasks. They mainly focused on the decomposition and independence of the tasks [2], [27]. However, none of them provided an effective method to address the collaboration between the workers. There are two forms of cooperation, policy-driven and informal cooperation. Policy-driven cooperation is done by exchange and correct handling of well-structured documents and concurrency control regarding the access to artifacts. Informal cooperation is characterized by the unrestricted exchange of structured or unstructured information [28]. In comparison, the latter has lower requirements for task publishers and workers. Because of the free interaction between workers, they understand each other better and collaborate together better [17]. Thus, it is more suitable for crowdsourced mode. At the spatial level, Internet-based crowdsourcing owns the advantaged environment to collaboration, and it is no need for workers to be concentrated in designated locations [29]–[30] At the time level, due to the uncertainty of the crowdsourcing groups, it is difficult to ensure a consistent working time. At the same time, most workers participated in crowdsourcing are amateur and can only be active online for crowdsourcing in certain periods. Therefore, the key to achieve collaboration in crowdsourcing software development is to select workers with the same active time as close as possible.

To ensure the communication between workers, and to improve the efficiency and quality of crowdsourcing software development, in this paper, based on the bidding model, we design a task assignment method for multi-modules software collaborative development from the perspectives of worker's ability, task module complexity and active time. Firstly, according to the total number of active time periods required by the task, we group the workers, and then optimize the global total utility using Hungarian algorithm, under the condition that the workers worked on the same task belong to the same collaborative candidate group. After that, using the replacement strategy of collaborative candidate group to solve the problem of assignment failure. Finally, all tasks in an assignment cycle are allocated.

This work makes the following contributions:

1) We propose the idea of assigning collaborative working groups through the coincidence of active time. Based on that, the task assignment problem of total utility optimization under the cooperative constraint is

proposed, and the formal definition of the problem is given.

2) We propose a series of algorithms to solve our problems. Our problem is transformed into a weighted bipartite graph optimal matching problem with special constraints. We propose an algorithm based on active time partitioning to make the working group satisfy the cooperative constraint. On the basis of the Hungarian algorithm, we propose a collaborative workgroup replacement strategy to optimize assignment.

3) Based on the real worker dataset, we provide experimental results of the proposed algorithm's performance on total utility and allocation success rate, which proves the effectiveness of our proposed solution.

II. CROWDSOURCING SOFTWARE TASK ASSIGNMENT

This section gives a definition of crowdsourcing software development mode and the problem faced.

A. CROWDSOURCING SOFTWARE DEVELOPMENT

In this paper, we mainly focus on studying a crowdsourcing software development system based on bidding mode. First, the publishers need to provide and publish task-related information to the crowdsourcing platform, from which developers then acquire the information of those available tasks. They select the competent and interested tasks to enroll. After that, the tasks whose registration ends and the number of applicants meets the requirements will be selected. For each selected task, workers who meet the requirements of collaborative work are divided into a group. Finally, selecting the most suitable worker for each module from the group. While completing the development tasks, each worker in the group submits the module code to the platform, and the platform will integrate all the codes to form a complete solution. Some relevant concepts in the context are defined as follows:

- *Crowdsourcing software development task*: the crowdsourcing software development task in this paper refers to the code programming task, divided into several smaller modules described by UML diagram[33]. It is described as a four tuple: $t = (M_t, c_t, g_t, R_t)$, where M_t is a set of task modules; c_t is the type of tasks; g_t is the time threshold that how long the workers are required to work together per day for task t; R_t is a group of workers who successfully registered for task t.
- *Task module*: the task module refers to the development module obtained by task decomposition. It is described as a two tuple:$m = (D_m, c_m)$, where D_m is the description of module m, including module requirements, rewards and other information. c_m is the expected complexity of module m.
- *Worker*: a worker is viewed as a participant in a crowdsourcing software development task, described as a two tuple: $d = (A_d, S_d)$, where $A_d = a_{d,1}, a_{d,2}, \ldots, a_{d,n}$ is the set of the ability of worker d for different types of tasks, so $a_{d,i}$ indicates the ability of worker d to complete the *ith* task; S_d is the active time set that workers are more inclined to carry out development work during these periods, $S_d \subseteq \{s \mid 1 \leq s \leq 24\}$, s represents the s hour of the 24 hours a day.
- *Assignment utility*: the assignment utility is an index to measure the quality of task assignment. It is generally believed that assigning harder modules to more capable workers as much as possible will generate higher value. The calculation of assignment utility of module m assigned to worker d is: $\mathrm{u}(m, d) = c_m a_{d,c_t}$.

B. DESCRIPTION OF THE PROBLEM

The problem of this paper is how to design a task assignment method to maximize the matching utility between workers and task modules in a multitasking context.

Assume that when the platform performing task assignment, the set of tasks satisfies the condition $T = t_1, t_2, \ldots t_n$, the final assignment result of task $t\,(t \in T)$ is $AR_t = \{(m_i, d_i) \mid i = 1, \ldots, |M_t|\}$,where tuple (m_i, d_i) indicates that module m_i was assigned to worker d_i. First of all, in order to meet the collaborative development requirements between task modules, the number of active sessions shared by workers is not less than g_t. Suppose $D_{AR_t} = \{d_i \mid (m_i, d_i) \in AR_t\}$, the shared active time is expressed as $TS = \bigcap_{d \in D_{AR_t}} S_d$ it must satisfy:

$$|\mathrm{TS}| = \left| \bigcap_{d_i \in D_{AR_t}} S_{d_i} \right| \geq g_t \tag{1}$$

The total assignment utility of a single task t is described as:

$$u_{AR_t} = \sum_{i=1}^{|M_t|} \mathrm{u}(m_i, d_i) \tag{2}$$

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

$$\begin{aligned} \max \; & \sum_{t \in T} u_{AR_t} \\ \text{s.t.} \; & d \in RD_t;\ \forall d \in D_{AR_t}, \forall \mathrm{t} \in \mathrm{T} \\ & \left| \bigcap_{d_i \in D_{AR_t}} S_{d_i} \right| \geq g_t;\ \forall \mathrm{t} \in \mathrm{T} \\ & D_{AR_i} \cap D_{AR_j} = \emptyset;\ \forall \mathrm{i} \in \mathrm{T}, \forall \mathrm{j} \in \mathrm{T}, \mathrm{i} \neq \mathrm{j} \end{aligned} \tag{3}$$

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

III. TASK ASSIGNMENT PROCESS

The specific framework of the crowdsourcing software task assignment method proposed in this paper is given in this

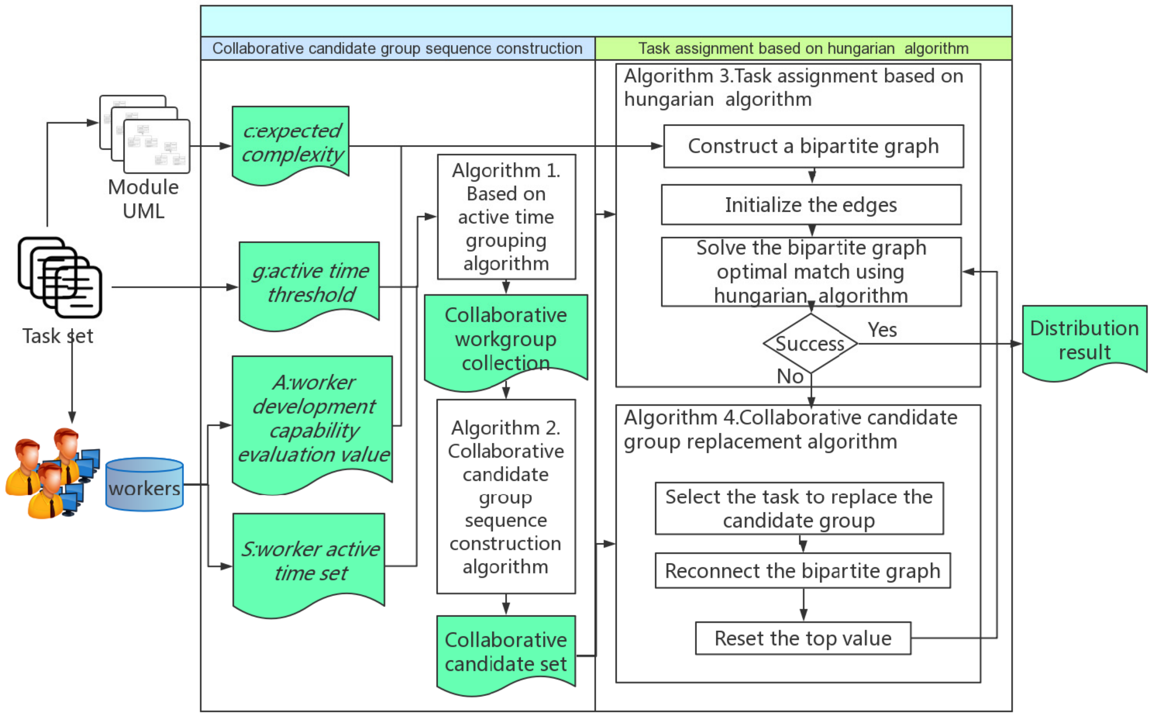

Collaborative candidate group sequence construction
Task assignment based on hungarian algorithm
Module UML
Task set
workers
c:expected complexity
g:active time threshold
A:worker development capability evaluation value
S:worker active time set
Algorithm 1. Based on active time grouping algorithm
Collaborative workgroup collection
Algorithm 2. Collaborative candidate group sequence construction algorithm
Collaborative candidate set
Algorithm 3.Task assignment based on hungarian algorithm
Construct a bipartite graph
Initialize the edges
Solve the bipartite graph optimal match using hungarian algorithm
Success
Yes
No
Distribution result
Algorithm 4.Collaborative candidate group replacement algorithm
Select the task to replace the candidate group
Reconnect the bipartite graph
Reset the top value

FIGURE 1. **The framework of crowdsourcing software development.**

section. The input of the assignment process is a set of assignable tasks, contain all the tasks required to meet the number of enrollees in the current assignment cycle. The output of the assignment process is the final assignment result, a set of module-worker pairs. The detailed assignment process is shown in Figure 1.

The assignment process can be divided into two steps:

(1) Constructing a collaborative candidate group sequence: In this step, for each input task, built a collaborative candidate group. Meanwhile, the registered workers for each task will be divided into multiple subsets in terms of the cooperative requirement. Then, the obtained subsets will be constructed as a specific priority sequence for subsequent matching.

(2) Optimizing matching results based on the Hungarian algorithm: In this part, a bipartite graph model will be constructed between task modules and workers. After initializing the links of the bipartite graph, the Hungarian algorithm is continuously used to solve the optimal matching scheme. When the matching fails, a replacement strategy of collaborative candidate group will be performed, and then rematch until it is successful or failure, due to the replacement strategy can no longer be executed.

IV. COLLABORATIVE CANDIDATE GROUP SEQUENCE CONSTRUCTION

In order to ensure that a group of workers assigned to the same task meet the specified simultaneous online requirements, it is necessary to let the crowd-workers have the same set of active time slots as far as possible. Therefore, we hold the threshold

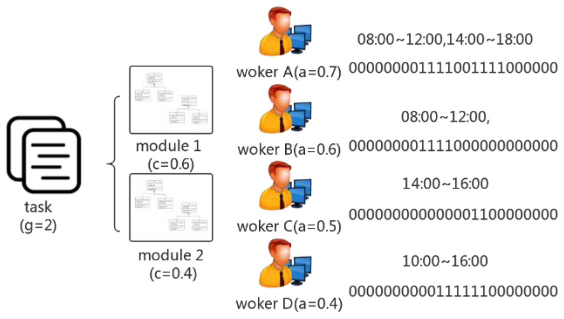


FIGURE 2. **An example of task and workers.**

of parameter g, and keep the total active time period owned by required workers is greater than or equal to g. As shown in Figure 2, 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, in the context: worker A and worker B have a total of 4 hours from 08:00 to 12:00, and they both meet the requirements; A and C have a total of 2 hours from 14:00 to 16:00, 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 these two time intervals can also meet the requirements; B and C have no common time period and do not meet the requirements. According to the requirements, B and D have a total of 2 hours from 10:00 to 12:00, and C and D have 2 hours from 14:00 to 16:00.

Therefore, 5 groups of workers meet the demand, {A, B}, {A, C}, {A, D}, {B, D}, {C, D}.

When the number of workers is large, we can't exhaust all the combinations that satisfy the conditions. Therefore, we divide the workers into multiple collaborative workgroups by grouping algorithm, according to whether the active time contains a specific g_t time period. The number of divisions is fixed, regardless of the number of workers. First, sort the workers in descending order according to their ability. Then, 24 sets of workers representing different time periods are generated. Each set contains all workers whose active time includes the time period they represent. Then, all the possible combinations of the g_t time periods are selected in the 24 time periods. At each iteration, calculate the intersection of the worker sets in each time period combination. If the number of workers in the intersection is greater than or equal to the number of the task modules $|M_t|$, 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 set $R_t$, Threshold $g_t$, Number of task
modules $|M_t|$
Output: Collaborative workgroup collection $CG_t = \{G_1, G_2, \ldots, G_n\}$
1.   Sort developers $R_t$ by ability $a_{d,c_t}$ of workers in the collection by
     descendant order
2.   Generate 24 worker sets $W_1, W_2, \ldots, W_{24}$
3.   for each $d \in R_t$ do
4.     for each $s \in S_d$ do
5.       $W_s \leftarrow W_s \cup \{d\}$;
6.     end for
7.   end for
8.   Generate all combinations of $g_t$ time periods from 24 time
     periods as $TM = \{P_1, P_2, \ldots, P_n\}$
9.   $CG_t \leftarrow \emptyset$;
10.  for each $P \in TM$ do
11.    $G \leftarrow R_t$;
12.    for each $s \in P$ do
13.      $G \leftarrow G \cap W_s$;
14.    end for
15.    if $|G| \geq |M_t|$ then
16.      $CG_t \leftarrow CG_t \cup G$;
17.  end for
18.  return $CG_t$;
```

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

Obviously, the assignment of task t must be one of the above-mentioned collaborative working groups or a subset, but with the number of workers becomes larger, the number of the subsets is also very large. In fact, any set of workers containing the optimal solution is equivalent to the assignment algorithm. Because the assignment algorithm will do its best to check the workers in the group, to avoid conflicts and ensure maximum assignment utility. For example, if the optimal assignment 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 assignment results of the best utility or the collaborative workgroup in which it is located. But the total utility of the assignment is limited by the workers' ability in the group. We can set the lower utility (the worst assignment effect that the group can assign to the task t) of the assigned workers, thereby limiting the total utility 's lower bound of the assignment. For a group of workers G, this lower bound is recorded as *worst* (G). In the assignment, the lower bound is gradually reduced until the assignment solution can be obtained, thereby ensuring that the total assignment utility is superior. The lower bound of the assignment utility is determined by the $|M|$ workers 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 $|M|$ 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 *worst* (G). 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 assignment 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, according to the collaborative workgroup in Table 2, four candidate collaborative groups can be got (see Table 3). Suppose that the rank of worker's ability is A>B>C>D, the available collaborative candidate groups are: ({A,B},{A,C},{A,D},{A,B,D},{A,C,D}).

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}

TABLE 3. The available collaborative candidate groups.

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}

Therefore, to generate a collaborative candidate sequence from a set of collaborative workgroups, a construction algorithm is proposed. First, initializing a min-heap *minHeap* of size k. Then obtaining the collaborative candidate groups $\{F_1, F_2, \ldots, F_{|G|-|M|}\}$ by intercepting a subset of sequence numbers $[1, |M|], [1, |M|+1] \ldots [1, |G|]$ for each group G in CG, and then for each candidate group F, calculate the worst assignment effect $worst\,(F) = \sum_{i=1}^{|M_t|} u(m_i, d_{|F|-|M_t|+i})$. The detailed steps are as follows:

1) The candidate group has been examined, skipped directly;

2) If the number of elements in the *minHeap* is less than k, then push F into *minHeap*;

3) When the number of elements in the *minHeap* is enough, if $worst\,(F)$ is greater than the worst assignment utility of the top candidate group, then the top group will be popped and the F will be pushed into the *minHeap*.

The construction algorithm is descripted as follows:

Algorithm 1 Collaborative Candidate Group Sequence Construction Algorithm

```
Input: Collaborative workgroup collection $CG_t$, Sequence size threshold $k$
Output: Collaborative candidate sequence $FG_t = \{F_1, F_2, \ldots, F_n\}$
1.  Initialize the min-heap $minHeap$ with a capacity of $k$;
2.  for each $G \in CG_t$ do
3.     for $i \leftarrow |M_t|$ to $|G|$ do
4.        Group workers with subscripts from 1 to $i$ in $G$ form collaborative candidate group $F$;
5.        if $F$ has not been investigated then
6.           if the size of $minHeap$ is less than $k$ then
7.              push $F$ into $minHeap$;
8.              Continue;
9.           $temp \leftarrow$ top group of $minHeap$;
10.          if $worst\,(F) < worst\,(temp)$ then
11.             push $F$ into $minHeap$;
12.          else
13.             Break;
14.    end for
15.  end for
16.  while $minHeap$ is not empty
17.     add the top group of $minHeap$ to the collaborative candidate sequence $FG_t$;
18.  return $FG_t$;
```

V. TASK ASSIGNMENT BASED ON HUNGARIAN ALGORITHM

The problem of crowdsourcing software task assignment is essentially equivalent to the matching between modules and crowdsourcing workers. Simply considering the goal of optimizing the total utility, the problem can be modeled as a maximum weight matching problem of bipartite graph. Hungarian algorithm [32], [31] is commonly used to handle this problem. Thus, in this paper, we attempt to design a novel assignment algorithm based on Hungarian algorithm. Hungarian algorithm is used to solve the optimal matching when the bipartite graph structure is determined. And the optimal matching with 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 optimize the bipartite graph matching using the Hungarian algorithm.

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

A. CONSTRUCT A BIPARTITE GRAPH FOR OPTIMAL MATCHING

After the registered workers of all tasks in one assignment period is grouped based on active time, set the left vertex set MV to the union of the module sets M_t of all tasks, and the right vertex set DV to the union of the optimal collaborative candidate groups. The weight of edges is viewed as the assign utility $u\,(m, d)$. Join all edges with weight and complete the construction of a bipartite graph. Then try to solve the bipartite graph optimal matching using the Hungarian algorithm. The process is outlined below:

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

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

c) Set the left and right set vertex edge weight to assign utility$u\,(m, d)$, connect the edges whose weight is not zero.

d) Try to use Hungarian algorithm to solve the bipartite graph optimal matching. If successful, 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:

B. REPLACEMENT STRATEGY FOR COLLABORATIVE CANDIDATE GROUP

The reason why the Hungarian 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 our research work is to build a sequence of collaborative candidate groups for each task before allocating. The algorithm allows changing the structure of the bipartite graph by replacing the collaborative candidate group based on the sequence. So, we can retry the conflict that the original bipartite graph could not handle. All collaborative candidate groups met the collaborative time constraint, and the replacement strategy used greedy thinking[34] to find the replacement will minimize the loss of the total assignment utility. The process is outlined below:

Algorithm 2 Task Assignment Algorithm Based on Hungarian Algorithm

```
input: task set $T$, Collaborative candidate sequence set $FG_t = \{F_1, F_2, \ldots, F_n\}$
output: assignment result
1.  initialize $U, V, E$;
2.  for each $t \in T$ do
3.    $F \leftarrow$ the top group in$FG_t$;
4.    for each $m \in M_t$ do
5.      $U \leftarrow U \cup \{vertex\ of\ m\}$;
6.      for each $d \in F$ do
7.        $V \leftarrow V \cup \{vertex\ of\ d\}$;
8.        connect the edges corresponding to m and d,
          and set the weight to$u(m, d)$;
9.      $E \leftarrow E \cup (m, d)$;
10.       end for
11.     end for
12.  end for
13.  using Hungarian algorithm to solve the optimal
     match of bipartite graph of $G = (U, V, E)$;
14.  if Match failed then
15.    Performing a collaborative candidate group
       replacement strategy;
16.    If Successful replacement then
17.      go back to step 16 to re-allocate;
18.    else
19.      assignment failed, end algorithm;
20.  return assignment result;
```

a) Calculate the worst-based utility difference between the current collaborative candidate group and the sub-optimal collaborative candidate group for each task involved, and select the task t_r with the smallest difference. Then set its current collaborative candidate group as the sub-optimal candidate group F_{t_r}.

b) Clear all edges of modules in taskt_r have been connected.

c) Connect each module in task t_r with workers in F_{t_r} one by one. Then re- execute the Hungarian algorithm initialization work of this partial bipartite graph.

The detailed algorithm is as follows:

VI. EXPERIMENT AND RESULT ANALYSIS

To verify the effectiveness of the proposed approach in this paper, we crawled the information of 28,834 workers from the programmer's inn. The worker's information includes work time and skill -level scores. In the experiment, the task data is generated by computer simulation. The number of modules per task is subject to the N (10, 1) distribution (Ignore negative values, the same below), and the complexity of the module is subject to the N (0.5, 1) distribution. Note that, we normalize the rating level as the development capability of the worker.

Algorithm 3 Collaborative Candidate Group Replacement Algorithm

```
input: Involved task set $T$, Collaborative candidate sequence set $\{FG_{t1}, FG_{t2}, \ldots, FG_{tm}\}$,current bipartite graph $G$
output: Replacement result
1.  $minGap \leftarrow inf$
2.  $t_r \leftarrow \emptyset$
3.  for each $t \in T$ do
1.    $F \leftarrow$ top element of $FG_t$
2.    if $F \neq \emptyset$ and worst $(F_t)$ −worst $(F) < minGap$
  then
3.      $t_r \leftarrow t$
4.  end for
5.  if $t_r \neq \emptyset$ then
6.    $F_{t_r} \leftarrow FG_{t_r}$pop the top element
7.    for each $m \in M_{t_r}$ do
8.      delete all existing $m$ related edges in $G$
9.      for each $d \in F_{t_r}$ do
10.       connect the edges corresponding to $m$ and $d$,
      set the weight to $u(m, d)$
11.     end for
12.   end for
13.  else
14.   replacement failed
```

The sequential assignment method is selected as the benchmark, which assigns each task randomly, selects the worker who meets the time constraint and assigns the best utility to match the task module. Workers who have been assigned will be excluded from the subsequent task assignments, to avoid multiple assignments.

A. COMPARISON OF TOTAL UTILITY

Compare with the single task priority assignment method (sequential assignment method) in terms of the total utility of the assignment.

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, respectively. 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 assignment method. The last column reflects the improvement ratio of the relative order assignment method of the algorithm in the total utility.

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

TABLE 4. Change the number of workers.

	task	modules	g	workers	proposed algorithm		sequential assignment	improvement
					execution time(s)	total utility	total utility	
ex1-1	20	117	3	150	6.37	29.352	23.382	25.53%
ex1-2	20	117	3	200	7.45	31.390	26.751	17.34%
ex1-3	20	117	3	250	9.56	33.016	29.069	13.58%
ex1-4	20	117	3	300	12.28	33.706	30.524	10.42%
ex1-5	20	117	3	350	14.79	34.710	31.648	9.68%
ex1-6	20	117	3	400	18.99	34.894	32.072	8.80%
ex1-7	20	117	3	450	24.55	36.026	33.481	7.60%
ex1-8	20	117	3	500	29.27	36.385	33.927	7.25%
ex1-9	20	117	3	550	38.65	36.691	34.274	7.05%
ex1-10	20	117	3	600	43.36	37.311	35.047	6.46%
average	20	117	3	375	20.52	34.348	31.018	11.37%

TABLE 5. Change the number of tasks.

	task	modules	g	workers	proposed algorithm		sequential assignment	improvement
					execution time (s)	total utility	total utility	
ex1-1	10	56	3	500	17.98	18.763	18.328	2.38%
ex1-2	15	94	3	500	23.81	31.356	30.389	3.18%
ex1-3	20	138	3	500	37.65	51.737	50.026	3.42%
ex1-4	25	144	3	500	50.82	51.352	49.416	3.92%
ex1-5	30	160	3	500	69.63	57.969	55.508	4.44%
ex1-6	35	199	3	500	142.30	59.658	57.454	3.84%
ex1-7	40	258	3	500	292.59	75.624	71.379	5.95%
ex1-8	45	263	3	500	292.12	91.251	85.075	7.26%
ex1-9	50	308	3	500	552.74	90.906	83.235	9.22%
ex1-10	55	334	3	500	819.42	90.535	85.218	6.24%
average	33	195	3	500	229.91	61.915	58.603	4.98%

TABLE 6. Change active time threshold.

	task	modules	g	workers	proposed algorithm		sequential assignment	improvement
					execution time(s)	total utility	total utility	
ex1-1	20	140	1	200	14.93	37.831	35.154	7.61%
ex1-2	20	140	2	200	22.01	37.588	34.965	7.50%
ex1-3	20	140	3	200	26.11	36.610	34.218	6.99%
ex1-4	20	140	4	200	28.29	35.598	32.655	9.01%
ex1-5	20	140	5	200	38.56	33.678	30.058	12.02%
ex1-6	20	140	6	200	46.96	33.384	29.780	12.10%
average	20	140	4	200	29.47	35.780	32.805	9.21%

According to the results of Table 5-7, in the case of various parameters, the proposed algorithm always outperforms the sequential assignment method in optimizing the total utility of assignment. As the parameters change, there are the following conclusions:

1) When other conditions remain unchanged, the smaller the number of workers participating in the assignment is, the higher the improvement ratio of the proposed algorithm is relatively. In Table 5, the improvement is up to 25%. It indicates that when the number of workers is small, the "waste"

TABLE 7. Change the number of workers.

	task	modules	g	workers	proposed algorithm success rate (%)	sequential assignment 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	88	27

phenomenon that may occur in the sequential assignment (the more powerful workers are assigned to the modules with lower complexity due to the order of assignment) will have a greater impact on the total utility. As the number of workers increases, the number of redundant workers with higher level increases, and even if there is a waste, there are fewer redundant workers to choose. Thus, the impact on the total utility is small. Using a Hungarian algorithm that accurately solves the optimal assignment of a particular bipartite graph, the proposed method in this paper is relatively stable and maintains a 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 assignment 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 assignment only focuses on the optimization of the single task, the impact on the subsequent assignment is generally negative, so the negative impact on the total utility is greater after multiple times of superposition.

3) When other conditions remain unchanged, the higher the active time threshold g, the higher the improvement ratio of the proposed algorithm. Table 7 shows that when the threshold g is increased, the total utility of both methods is reduced. Because the collaborative conditions become harsh, it is more difficult to ensure the selected workgroup's ability at a higher level. For the sequential assignment, the effect of "waste" is more serious when fewer workers satisfy the synergistic constraint, and resulting in a lower overall utility of the final assignment.

B. ASSIGNMENT SUCCESS RATE COMPARISON

This experiment compares the assignment success rate between the proposed method and the single task priority assignment method (sequential assignment 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 assignment success rates of the proposed algorithm and the sequential assignment 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 assignment. 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 obeys $N(0.5, 1)$. 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 assignment redundant.

From the results of Table 8-10, the advantages of the proposed algorithm are very obvious in the assignment 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 assignment 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 assignment success rate of the two groups is very low, the success rate

TABLE 8. Change the number of tasks.

	task	modules	g	workers	proposed algorithm	sequential assignment
					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	87	16

TABLE 9. Change the active time threshold.

	task	modules	g	workers	proposed algorithm	sequential assignment
					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	54	23

of the sequential assignment 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 assignment requires more redundant workers to achieve a higher assignment 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 assignment success rate increases. However, the sequential assignment 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 assignment success rate.

3) When the threshold g is increased, the assignment 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 assignment 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 assignment 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 assignment success rate decreases. Since the method can save by modifying the allocated scheme when the single task assignment fails, and the sequential assignment method has no corresponding measures, so the sequential assignment method is more affected when the threshold g is increased.

VII. CONCLUSION

This paper models the collaborative software task assignment problem in the crowdsourced environment as the assignment optimization problem, and integrates the three factors of worker capacity, task module complexity and worker active time to establish optimization goals. Based on the Hungarian 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 assignment method.

However, the research in this paper still has limitations. In this paper, the definition of collaboration in software crowdsourcing is relatively simple. In the actual development process, the required level of collaboration between workers will be differentiated by inter-module dependencies. So, the definition of collaboration constraints will be more complicated. In addition, the allocation utility calculation method set in this paper only considers the development ability of the workers and the complexity of the module. But there are many other factors that might have the impact on task assignment. such as the character of the worker, the attitude, the communication ability, etc. Although the proposed algorithm can be applied to more complicated utility calculation methods. But it is necessary to consider how to set a reasonable calculation formula.

In fact, it is difficult to determine the pros and cons of the task assignment [35], [36]. The utility of the proposed method is based on the accuracy of the worker-task matching measurement. it means that it is important to have an accurate judgment on the ability of the worker and the complexity of the module. Therefore, the future research will be carried out on the measurement of the ability of the worker and the module complexity.

REFERENCES

[1] A. Begel, J. Bosch, and M.-A. Storey, "Social networking meets software development: Perspectives from GitHub, MSDN, stack exchange, and topcoder," *IEEE Softw.*, vol. 30, no. 1, pp. 52–66, Jan. 2013.

[2] K.-J. Stol and B. Fitzgerald, "Two's company, three's a crowd: A case study of crowdsourcing software development," in *Proc. 36th Int. Conf. Softw. Eng.*, May 2014, pp. 187–198.

[3] B. Fitzgerald, "Crowdsourcing software development: Silver bullet or lead balloon," in *Proc. 5th Int. Workshop Artif. Intell. Requirements Eng. (AIRE)*, Aug. 2018, pp. 29–30.

[4] K.-J. Stol, B. Caglayan, and B. Fitzgerald, "Competition-based crowdsourcing software development: A multi-method study from a customer perspective," *IEEE Trans. Softw. Eng.*, vol. 45, no. 3, pp. 237–260, Mar. 2017.

[5] K.-J. Stol and B. Fitzgerald, "Researching crowdsourcing software development: Perspectives and concerns," in *Proc. 1st Int. Workshop CrowdSourcing Softw. Eng.* New Yor, NY, USA, Jun. 2014, pp. 7–10.

[6] X. Peng, M. Ali Babar, and C. Ebert, "Collaborative software development platforms for crowdsourcing," *IEEE Softw.*, vol. 31, no. 2, pp. 30–36, Mar. 2014.

[7] A. Dwarakanath *et al.*, "Crowd build: A methodology for enterprise software development using crowdsourcing," in *Proc. IEEE/ACM 2nd Int. Workshop CrowdSourcing Softw. Eng.*, May 2015, pp. 8–14.

[8] H. Tajedin and D. Nevo, "Determinants of success in crowdsourcing software development," in *Proc. Annu. Conf. Comput. People Res.*, May 2013, pp. 173–178.

[9] L. Machado, J. Kroll, S. Marczak, and R. Prikladnicki, "Breaking collaboration barriers through communication practices in software crowdsourcing," in *Proc. IEEE 11th Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2016, pp. 44–48.

[10] J. E.Tomayko and O. Hazzan, *Human Aspects of Software Engineering* (Electrical and Computer Engineering Series). Rockland, MA, USA: Charles River Media, 2004.

[11] A. L. Zanatta, I. Steinmacher, L. S. Machado, C. R. B. de Souza, and R. Prikladnicki, "Barriers faced by newcomers to software-crowdsourcing projects," *IEEE Softw.*, vol. 34, no. 2, pp. 37–43, Mar./Apr. 2017.

[12] S. Bandinelli, E. D. Nitto, and A. Fuggetta "Supporting cooperation in the SPADE-1 environment," *IEEE Trans. Softw. Eng.*, vol. 22, no. 12, pp. 841–865, Dec. 2002.

[13] K. R.Lakhani, D. A. Garvin, E. Lonstein, "Topcoder (a): Developing software through crowdsourcing," *Harvard Bus. School Gen. Manage. Unit Case*, pp. 610–632, 2010.

[14] Y. Fu, H. Chen, and F. Song, "STWM: A solution to self-adaptive task-worker matching in software crowdsourcing," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.* Cham, Switzerland: Springer, 2015, pp. 383–398.

[15] A. L. Zanatta, L. S. Machado, G. B. Pereira, R. Prikladnicki, and E. Carmel, "Software crowdsourcing platforms," *IEEE Softw.*, vol. 33, no. 6, pp. 112–116, Nov./Dec. 2016.

[16] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," *J. Syst. Softw.*, vol. 126, pp. 57–84, Apr. 2017.

[17] Z. Shi *et al.*, "Task allocation mechanism for crowdsourcing system based on reliability of users," *J. Comput. Appl.*, vol. 37, no. 9, pp. 2449–2453, 2017.

[18] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman, "Developer recommendation for crowdsourced software development tasks," in *Proc. IEEE Symp. Service-Oriented Syst. Eng.*, Mar./Apr. 2015, pp. 347–356.

[19] W. SHAO, X. WANG, and W. JIAO, "A developer recommendation framework in software crowdsourcing development," in *Software Engineering and Methodology for Emerging Domains*. Singapore: Springer, 2016.

[20] J. Zhu, B. Shen, and F. Hu, "A learning to rank framework for developer recommendation in software crowdsourcing," in *Proc. Asia-Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2015, pp. 285–292.

[21] Z. Wang, H. Sun, Y. Fu, and L. Ye, "Recommending crowdsourced software developers in consideration of skill improvement," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng.* Washington, DC, USA, Oct./Nov. 2017, pp. 717–722.

[22] L. Machado, R. Prikladnicki, F. Meneguzzi, C. R. B. de Souza, and E. Carmel, "Task allocation for crowdsourcing using AI planning," in *Proc. IEEE/ACM 3rd Int. Workshop CrowdSourcing Softw. Eng. (CSI-SE)*, May 2016, pp. 36–40.

[23] S. Lee, S. Park, and S. Park, "A quality enhancement of crowdsourcing based on quality evaluation and user-level task assignment framework," in *Proc. Int. Conf. Big Data Smart Comput. (BIGCOMP)*, Jan. 2014, pp. 60–65.

[24] S. B. Roy, I. Lykourentzou, S. Thirumuruganathan, S. Amer-Yahia, and G. Das, "Task assignment optimization in knowledge-intensive crowdsourcing," *Int. J. Very Large Data Bases*, vol. 24, no. 4, pp. 467–491, Aug. 2015.

[25] I. Boutsis and V. Kalogeraki, "On task assignment for real-time reliable crowdsourcing," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, Jun./Jul. 2014, pp. 1–10.

[26] H. Rahman, S. B. Roy, S. Thirumuruganathan, S. Amer-Yahia, and G. Das, "Task assignment optimization in collaborative crowdsourcing," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2015, pp. 949–954.

[27] K. Li, J. Xiao, Y. Wang, and Q. Wang, "Analysis of the key factors for software quality in crowdsourcing development: An empirical study on topcoder.com," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf.*, Jul. 2013, pp. 812–817.

[28] W. R. Bischofberger, T. Kofler, K.-U. Matzel, and B. Schaffer, "Computer supported cooperative software engineering with Beyond-Sniff," in *Proc. Softw. Eng. Environ.*, Apr. 1995, pp. 135–143.

[29] T. D. LaToza and A. van der Hoek, "Crowdsourcing in software engineering: Models, motivations, and challenges," *IEEE Softw.*, vol. 33, no. 1, pp. 74–80, Jan./Feb. 2016.

[30] N. Leicht *et al.*, "Crowdsourcing in software development: A state-of-the-art analysis," in *Proc. 28th Bled eConference #eWellbeing*, Bled, Slowenien, 2015. [Online]. Available: https://www.alexandria.unisg.ch/243336/

[31] J. Munkres, "Algorithms for the assignment and transportation problems," *J. Soc. Ind. Appl. Math.*, vol. 5, no. 1, pp. 32–38, Mar. 1957.

[32] H. W. Kuhn, "The Hungarian method for the assignment problem," *Nav. Res. Logistics*, vol. 2, nos. 1–2, pp. 83–97, Mar. 2010.

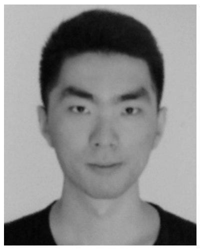
[33] C. F. J. Lange, M. R. V. Chaudron, and J. Muskens, "In practice: UML software architecture and design description," *IEEE Softw.*, vol. 23, no. 2, pp. 40–46, Mar. 2006.

[34] P. E. Black, "Greedy algorithm," in *Dictionary of Algorithms and Data Structures*. Feb. 2005. [Online]. Available: https://www.nist.gov/dads/HTML/greedyalgo.html

[35] L. B. Chilton, J. J. Horton, J. J. Horton, and S. Azenkot, "Task search in a human computation market," in *Proc. ACM SIGKDD Workshop Hum. Comput. ACM*, Jul. 2010, pp. 1–9.

[36] E. Aldhahri, V. Shandilya, and S. Shiva, "Towards an effective crowdsourcing recommendation system: A survey of the state-of-the-art," in *Proc. IEEE Symp. Service-Oriented Syst. Eng.*, Mar./Apr. 2015, pp. 372–377.

DUNHUI YU received the M.S. degree from Hubei University, Hubei, China, and the Ph.D. degree from the Wuhan University, Hubei. He is currently the Director of the Software Engineering Department, School of Computer and Information Engineering, Hubei University, and the Deputy Director of the Hubei Engineering Research Center of Educational Informationalization. In the past five years, he has hosted several projects, including the Natural Science Foundation of Hubei Province and the Education Department of Hubei Province. He has participated in many projects, such as 973, 863, and the National Natural Science Foundation. He presided over a number of horizontal issues, such as the four-in-one performance appraisal system for Guizhou local taxes, with a total investment of more than 3.5 million Yuan. He has published more than 20 scientific papers in core journals and conferences. He participated in the preparation of an academic monograph, and applied for invention patents. He is a member of the China Computer Application Committee and the Chinese Computer Society. He was a recipient of the Wuhan Excellent Academic Paper Award.



ZHUANG ZHOU received the B.S. degree from Jiangxi Normal University, in 2016. He is currently pursuing the M.S. degree in computer technology from Hubei University, Hubei, China. His research interest includes crowdsourcing software development.

YI WANG received the B.S. degree from Hubei Polytechnic University, in 2016. He is currently pursuing the M.S. degree in computer application technology from Hubei University, Hubei, China. His research interest includes crowdsourcing.