Information Sciences 586 (2022) 119–139

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Multi-stage complex task assignment in spatial crowdsourcing

Zhao Liu [a], Kenli Li [a,*], Xu Zhou [a], Ningbo Zhu [a], Yunjun Gao [b], Keqin Li [a,c]

[a] College of Information Science and Engineering, Hunan University, changsha, China
[b] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[c] Department of Computer Science, State University of New York, New Paltz, New York, USA

ARTICLE INFO

Article history:
Received 21 August 2021
Received in revised form 27 November 2021
Accepted 28 November 2021
Available online 2 December 2021

Keywords:
Spatial data
Crowdsourcing
Multi-stage
Dependency
Task assignment

ABSTRACT

With the widespread application of smart devices, spatial crowdsourcing (SC) has been extensively integrated into daily life. Task assignment is a crucial issue in SC and has attracted much attention. Most prior studies on task assignment ignore the importance of dependency among tasks, resulting in some ineffective matching pairs and wasting workers' time. To this end, we formulate a new problem in SC, abbreviated as multi-stage complex task assignment (MSCTA), which aims to assign workers to multi-stage complex tasks to maximize the total profit. Compared with existing studies, MSCTA can obtain more effective assignments by considering the dependency constraints among tasks. We prove that the MSCTA problem is NP-hard and propose a greedy algorithm and a game algorithm. Specifically, both algorithms iteratively utilize a filtering module to obtain a set of executable tasks (ET) for assignment. The greedy algorithm can quickly assign the most profitable workers to the subtasks in each round of ET, and obtain a provable approximate result. The game algorithm is proved to be convergent and can win a Nash equilibrium when processing the subtasks in each round of ET. Extensive experimental results demonstrate the efficiency of our algorithm.

1. Introduction

With the development and widespread use of smart devices and 5G, spatial crowdsourcing [1] has been widely used in various applications, such as Meituan, DidiChuxing, and Taskrabbit [2–4]. In spatial crowdsourcing, crowdsourcing platforms collect spatial tasks and require workers to move to specific locations to complete the assigned tasks.

In terms of task complexity, researches on spatial crowdsourcing can be divided into two categories. First, tasks such as delivering food, monitoring traffic conditions, and measuring network quality are simple and easy to complete [5–7]. Second, tasks are complex, e.g., repairing a house, holding an orienteering race, and preparing for a party. These tasks consist of multiple stages and require the cooperation of several skilled workers [8–10].

For the complex-task assignment problem in spatial crowdsourcing, existing studies [11] usually consider the constraints of skills, time, distance, and budget. In [12], Cheng et al. presented a *g-D&C* algorithm to match each complex task with a group of workers that meet the specified constraints. However, in many applications, complex tasks are composed of multiple subtasks (or stages) with dependency constraints, meaning that a subtask cannot be processed until its dependent subtasks have been completed. For example, repairing a house is a multi-stage complex task, including the following steps: first,

* Corresponding author.
E-mail addresses: lzhao@hnu.edu.cn (Z. Liu), lkl@hnu.edu.cn (K. Li), zhxu@hnu.edu.cn (X. Zhou), quietwave@hnu.edu.cn (N. Zhu), gaoyj@hnu.edu.cn (Y. Gao), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.ins.2021.11.084
0020-0255/

repairing the floor and plumbing system, second, installing electrical circuits and painting the walls, finally, furnishing and cleaning. Another example is delivering packages to some strictly managed residential areas during the COVID-19 outbreak: first, transporting packages to the designated locations by some workers with driving skills, second, delivering the packages to customers by other workers with medical protection capabilities. Last example, holding a ceremony can also be regarded as a multi-stage task, each stage requires workers with different skills to work together.

For these multi-stage complex tasks aforementioned, if the dependency constraints are not considered when solving the task assignment problem, they always obtain invalid assignments. In addition, each subtask has a deadline, and the workers assigned to each subtask must arrive at the location on time. Moreover, each complex multi-skilled subtask requires the cooperation of workers, making the task assignment problem more complicated. Motivated by these issues, we attempt to investigate a new spatial crowdsourcing problem, called multi-stage complex task assignment (MSCTA). The purpose of MSCTA is to assign workers to multi-stage complex tasks and maximize the total profit under the constraints of dependency, maximal working area, skills, budget, and deadline.

Next, we illustrate the MSCTA problem in Example 1.1.

Example 1. (Holding a Wedding.) Fig. 1 shows an application of spatial crowdsourcing, that is, holding a wedding. This application contains many complex tasks, such as decorating the wedding scene and wedding cars, purchasing, playing music, taking photos, providing catering services, and offering entertainment. These tasks belong to different stages of a wedding ceremony.

Therefore, holding a wedding can be regarded as a multi-stage task t_j. Without loss of generality, task t_j contains three dependent subtasks (stages), namely $t_{j,1}, t_{j,2}$, and $t_{j,3}$. These three subtasks represent before, during, and after the wedding ceremony, respectively. Furthermore, each subtask requires the cooperation of workers with different skills to complete.

In Fig. 1, there are two multi-stage tasks $(t_1 - t_2)$ and six multi-skilled workers $(w_1 - w_6)$, see Table 1 for details. As shown in Fig. 1, each dotted line with an arrow starts from a subtask and points to its dependent subtasks. The purpose of the crowdsourcing platform is to assign workers to multi-stage complex tasks to maximize total profit under the constraints of dependency, maximal working distance, skills, budget, and deadline.

As shown in Fig. 1(a), workers are assigned to subtasks without considering dependency constraints. Specifically, the assignments of workers are shown by solid arrows and presented in Table 2. We obtain that for task t_1, workers w_1 and w_2 can provide the skills required by subtasks $t_{1,2}$ and $t_{1,3}$, respectively, but w_3 cannot provide the skills required by $t_{1,1}$. According to the dependency constraint, subtask $t_{1,2}$ depends on $t_{1,1}$, and $t_{1,3}$ depends on $t_{1,2}$. Therefore, w_1 and w_2 must wait until $t_{1,1}$ is completed; that is to say, the matching pairs $\langle t_{1,2}, w_1\rangle$ and $\langle t_{1,3}, w_2\rangle$ are invalid. Similarly, for task t_2, the matching pairs $\langle t_{2,2}, w_4\rangle, \langle t_{2,3}, w_5\rangle$ and $\langle t_{2,3}, w_6\rangle$ are also invalid. As a result, no subtasks can be completed due to the neglect of the dependency constraints. Based on the dependency constraints among the subtasks, we get the assignments shown in Fig. 1(b). For task t_1, since the skills required by $t_{1,1}, t_{1,2}$ and $t_{1,3}$ are covered by the assigned workers, the matching pairs $\langle t_{1,1}, w_3\rangle, \langle t_{1,1}, w_6\rangle, \langle t_{1,2}, w_1\rangle$, and $\langle t_{1,3}, w_2\rangle$ are valid. Similarly, for task t_2, the matching pairs $\langle t_{2,1}, w_4\rangle$ and $\langle t_{2,1}, w_5\rangle$ are also valid. Based on the assignments presented in Table 2, subtasks $t_{1,1}, t_{1,2}, t_{1,3}$ and $t_{2,1}$ can be completed.

To solve the MSCTA problem, we need take into account the following points: 1) subtasks contained in the same multi-stage task are complex and dependent on each other; and 2) the total profit is affected by two aspects, the profit achieved from each subtask and the number of completed subtasks.

The MSCTA problem is most related to the dependency-aware spatial crowdsourcing (DA-SC) problem in [13], considering the dependencies between tasks. However, the tasks in [13] are single-skilled and can be accomplished by one worker, while many tasks in real life are multi-skilled and require the cooperation of multi-skilled workers. In this paper, our methods can be used to handle single-skilled tasks as well as multi-skilled tasks. In addition, when solving the DA-SC problem

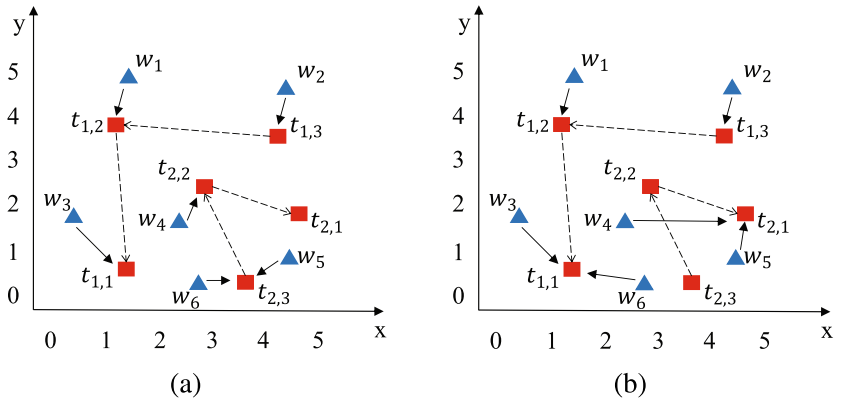

Fig. 1. An example of holding a wedding.

Table 1
Details of tasks and workers.

(a) Task

Task	Subtasks	Skills	Dependency
t_1	$t_{1,1}$	s_1, s_2	$\varnothing$
	$t_{1,2}$	s_2, s_3	$t_{1,1}$
	$t_{1,3}$	s_1, s_3	$t_{1,1}, t_{1,2}$
t_2	$t_{2,1}$	s_1, s_2, s_3	$\varnothing$
	$t_{2,2}$	s_1	$t_{2,1}$
	$t_{2,3}$	s_2, s_3	$t_{2,1}, t_{2,2}$

(b) Workers

Worker	Skills
w_1	s_2, s_3
w_2	s_1, s_3
w_3	s_1
w_4	s_1, s_2
w_5	s_1, s_3
w_6	s_2

Table 2
Task Assignment Schemes.

Task	Stage (Subtask)	Fig. 1(a)	Fig. 1(b)
t_1	$t_{1,1}$	w_3	w_3, w_6
	$t_{1,2}$	w_1	w_1
	$t_{1,3}$	w_2	w_2
t_2	$t_{2,1}$	$\varnothing$	w_4, w_5
	$t_{2,2}$	w_4	$\varnothing$
	$t_{2,3}$	w_5, w_6	$\varnothing$

[13], Ni et al. first combined multiple tasks with dependencies into an associative task set, and then treated the associative task set as a complex task and assigned a group of workers to complete. However, if a task in the associative task set does not find a worker satisfying the constraints, all tasks in the associative set cannot be assigned, thus affecting the completion rate. Moreover, different from our work, the purpose of DA-SC is to maximize the number of assigned task-and-worker pairs.

In this paper, we first formulate the multi-stage complex task assignment (MSCTA) problem and prove it is NP-hard. Then, we propose two algorithms, namely the greedy algorithm and the game algorithm, to solve the MSCTA problem effectively. Generally, the contributions of this paper can be summarized as follows:

- We formulate a multi-stage complex task assignment (MSCTA) problem and demonstrate that it is NP-hard in Section 3.
- We present a greedy algorithm to handle the MSCTA problem in Section 5. We analyze the time complexity of the algorithm, and prove the approximate boundary of the total profit $V(A_p)$ obtained by the algorithm.
- We design a game algorithm to solve the MSCTA problem in Section 6. We analyze the algorithm from three aspects: the stability, the convergence rate, and the quality of the obtained result.
- We conduct multiple experiments on real data sets and synthetic data sets to demonstrate the effectiveness of our proposed algorithms in Section 7.

In addition, we introduce related work in Section 2 and present a framework to address the MSCTA problem in Section 4. Finally, we summarize this paper in Section 8.

2. Related work

Spatial crowdsourcing is a new framework in crowdsourcing [14–16], requiring workers to physically move from their original positions to the locations of tasks. With the widespread of smart devices, spatial crowdsourcing has been widely used in academia and industry, such as Meituan [2], DidiChuxing [3] and TaskRabbit [4]. As the basic challenge in spatial crowdsourcing [1], task assignment can be divided into the following categories according to the arrival scenarios and the algorithmic assignment model [17]: static matching [18], static planning [19], dynamic matching [20] and dynamic planning [21]. In the matching model, task assignment is often formulated as a bipartite graph-based problem: workers and tasks can be represented by the vertices in the bipartite graph, the utility or cost between a worker and a task can be denoted by the weight of an edge, and the problem is to obtain an optimal matching in the bipartite graph. In the planning model (a.k.a. scheduling model), task assignment aims to plan a route for each worker to perform a sequence of tasks. In addition, accord-

ing to the publishing models mentioned in [22], task assignment can be classified into two other categories: 1) in the worker-selected tasks (WST) mode, workers can choose tasks according to their preferences [23,24]; and 2) in server-assigned tasks (SAT) mode, tasks will be assigned by the crowdsourcing platform according to the basic attributes of workers and tasks [25]. In this paper, the MSCTA problem belongs to the category of dynamic matching, we first introduce a batch-based framework, and then propose a greedy method based on the SAT mode and design a game method based on the WST mode.

In many spatial crowdsourcing applications, tasks are simple and can be accomplished by ordinary workers, such as taking photos [6] and delivering food [5,26]. Besides these simple tasks, many spatial tasks are complex and require a cooperation effort of a set of skilled workers to complete [8–11], such as holding a party and repairing a house. For the assignment of complex tasks, existing studies either assign a group of workers to a complex task that meets the skill constraints or treats a complex task as a combination of multiple simple subtasks and assign a skilled worker to each subtask [13]. Most of these studies do not consider the dependency constraints among tasks, leading to many invalid assignments.

In addition, various optimization goals have been discussed in previous works. Zhao et al. [27] aimed to accomplish as many tasks as possible, and in [28], they studied how to reduce the total cost of workers while keeping the number of completed tasks unchanged. In [29], Xu et al. proposed an insertion operator for dynamic ride-sharing and aimed to minimize the maximum flow time of requests or reduce the total travel time of workers. In [30], Cheng et al. considered the cooperative relationship between workers and aimed to maximize the total cooperation quality revenue. In this paper, the MSCTA problem aims to assign workers to multi-stage complex tasks for gaining the obtained total profit. Even though the problem of maximizing the total score of the system has been studied [12], the dependency constraints between tasks have not been considered.

The multi-objective optimization problems in spatial crowdsourcing have also been investigated in prior studies. Wang et al. [31] investigated a problem of heterogeneous spatial crowdsourcing task allocation, which proposed a solution to maximize the assigned task coverage and minimize the incentive cost simultaneously. In [32], Xiao et al. introduced a novel spatial crowdsourcing task assignment problem, called competitive detour tasking, and proposed a light-weight secure reverse auction mechanism to assign tasks which can protect workers' private sensitive information perfectly. In [33], Zheng et al. studied the problem of multi-campaign oriented spatial crowdsourcing, which aimed to maximize the objective function comprised by the score terms of the throughput, distance, and the worker diversity of the campaigns. Although some existing multi-objective optimization papers studied the same objective function as ours, they did not consider the constraints of multi-skills and dependencies of tasks.

3. Problem statement

In this section, we present the problem of multi-stage complex task assignment in spatial crowdsourcing. Table 3 lists the commonly used notations in this paper.

Definition 1 (*Multi-skilled Workers*). At timestamp p, the platform collects a set of workers $W_p = \{w_1, w_2, \cdots, w_n\}$. A worker $w_i \in W_p$ connects to the crowdsourcing platform at timestamp a_i and reports his/her location $l_i = (l_i(x), l_i(y))$. Worker w_i has a set of skills sk_i, and can move in any direction at speed v_i with the maximum moving distance d_i. We denote worker w_i as $w_i = (l_i, sk_i, d_i, v_i, a_i)$.

According to Definition 1, worker w_i can perform some skilled work based on the skills sk_i, such as decorating houses and repairing electrical appliances. Additionally, workers can log in and log out of the platform freely, and they can choose tasks according to their preferences (e.g., selecting the closest spatial tasks or tasks with high profits) or accept tasks assigned by

Table 3
Notations and Explanations.

Notation	Explanation
w_i, W_p	a skilled worker and a set of workers collected at timestamp p, respectively
d_i, v_i, l_i	the maximum moving distance of worker w_i, the moving speed of worker w_i and the location of worker w_i, respectively
$t_j, t_{j,k}, T_p$	a multi-stage complex task and the kth subtask (stage) in task t_j, and a set of tasks collected at timestamp p, respectively
$sk_i, sk_{j,k}$	the skill set of worker w_i and the skill set required by subtask $t_{t,k}$, respectively
a_i, a_j	the timestamp of worker w_i appearing on the platform and the timestamp of task t_j appearing on the platform
$wt_{j,k}, l_{j,k}$	the deadline to start processing subtask $t_{j,k}$ and the location of subtask $t_{j,k}$, respectively
$b_{j,k}, D_{j,k}$	the budget provided by subtask $t_{j,k}$ and the dependent task set of $t_{j,k}$, respectively
$VC_{j,k}$	a set of valid candidate workers for subtask $t_{j,k}$
ET, A_p	a set of executable subtasks and an assignment result, respectively
$A_p(t_{j,k})$	a set of task-worker matching pairs for subtask $t_{j,k}$
$\|s(A_p(t_{j,k}))\|$	the number of required skills provided by the matched workers in $A_p(t_{j,k})$
$C(A_p(t_{j,k}))$	the sum of the travel cost of workers in $A_p(t_{j,k})$ to the location of subtask $t_{j,k}$
CT_p	a set of subtasks that meet the completion conditions
$c_{i,j,k}$	the travel expenses of worker w_i to the location of subtask $t_{j,k}$
$\langle t_{j,k}, w_i \rangle$	a valid task-worker matching pair

the platform. In the model of workers accepting tasks assigned by the platform, the preferences of workers are not considered in the process of task assignment, and the platform assigns tasks to workers based on the principle of maximizing the global objective function. Each worker can only match one subtask in a batch, after the worker completes the currently assigned subtask, s/he can be collected by the platform again and participate in the processing of the next batch of tasks.

Definition 2 (*Multi-stage Complex Spatial Tasks*). Let $T_p = \{t_1, t_2, \cdots, t_m\}$ be a set of multi-stage complex tasks, collected by the platform at timestamp p. Task t_j $(1 \leqslant j \leqslant m)$ consists of l subtasks (or l stages), $t_j = \{t_{j,1}, t_{j,2}, \cdots, t_{j,l}\}$, and appears on the platform at timestamp a_j. Each subtask $t_{j,k}$ $(1 \leqslant k \leqslant l)$ reports its location $l_{j,k} = (l_{j,k}(x), l_{j,k}(y))$ and must start being processed within $wt_{j,k}$. Subtask $t_{j,k}$ can only be accomplished by a set of workers with the required skills $sk_{j,k}$. In addition, subtask $t_{j,k}$ provides a budget $b_{j,k}$ to motivate workers to complete it. Moreover, the subtasks contained in a task are dependent, and $t_{j,k}$ depends on a set of subtasks $D_{j,k}$. Therefore, subtask $t_{j,k}$ can be denoted as $t_{j,k} = (l_{j,k}, sk_{j,k}, wt_{j,k}, b_{j,k}, a_j, D_{j,k})$.

In spatial crowdsourcing, workers need to physically move from their locations to the locations of subtasks and spend time processing. Note that, the subtasks in the same complex task are posted at the same time, but need to be started at different times in order. In addition, the locations of these subtasks can be identical or different. To mobilize the enthusiasm of workers, each subtask $t_{j,k}$ provides a budget $b_{j,k}$ to compensate for the consumption of workers. In Definition 2, the subtasks contained in a task are dependent. For example, if task t_j consists of three subtasks $t_j = \{t_{j,1}, t_{j,2}, t_{j,3}\}$, where $t_{j,2}$ depends on $t_{j,1}$, and $t_{j,3}$ depends on $t_{j,2}$, that is, $D_{j,2} = \{t_{j,1}\}$ and $D_{j,3} = \{t_{j,1}, t_{j,2}\}$. Therefore, we can obtain that subtask $t_{j,2}$ can only be conducted after subtask $t_{j,1}$ is completed, and subtask $t_{j,3}$ can only be executed after $t_{j,1}$ and $t_{j,2}$ are completed. In other words, if subtask $t_{j,1}$ meets the completion conditions, then $D_{j,2} = \varnothing$. Similarly, if both subtasks $t_{j,1}$ and $t_{j,2}$ meet the completion conditions, then $D_{j,3} = \varnothing$.

Definition 3 (*Valid Matching Pair*). After the crowdsourcing platform collects a batch of workers W_p and tasks T_p at timestamp p, it performs valid task-worker matching. Since each task t_j $(1 \leqslant j \leqslant m)$ consists of l subtasks (or l stages), a valid task-worker matching pair $\langle t_{j,k}, w_i \rangle$ means that the matching pair simultaneously satisfies the constraints of dependence, maximal working area, skills, budget, and deadline.

According to Definition 3, a valid task-and-worker matching pair $\langle t_{j,k}, w_i \rangle$ needs to satisfy the following conditions: 1) the distance between the location l_i of worker w_i and the location $l_{j,k}$ of subtask $t_{j,k}$ must be less than the maximum moving distance d_i; 2) worker w_i can reach the location $l_{j,k}$ of subtask $t_{j,k}$ before the deadline $(a_j + wt_{j,k})$; 3) worker w_i can provide some skills required by subtask $t_{j,k}$, that is, $sk_i \cap sk_{j,k} \neq \varnothing$; 4) the budget $b_{j,k}$ provided by subtask $t_{j,k}$ should at least be able to compensate the travel expenses of workers; and 5) the dependency constraint of subtask $t_{j,k}$ must be satisfied, that is, if a subtask $t_{j,k}$ depends on a subtask $t_{j,k'}$, then $t_{j,k}$ can only be conducted after the dependent subtask $t_{j,k'}$ has been matched enough workers, and these workers can reach the location of $t_{j,k'}$ on time and provide the required skills.

For all matching pairs containing subtask $t_{j,k}$, if these matching pairs are valid and the required skills of subtask $t_{j,k}$ can be fully covered by the assigned workers, then we claim that subtask $t_{j,k}$ meets the *completion condition*.

Before introducing the MSCTA problem, we would like to discuss the profit obtained from subtask $t_{j,k}$. Suppose that, subtask $t_{j,k}$ has matched a set of valid task-and-worker matching pairs $A_p(t_{j,k})$.

If subtask $t_{j,k}$ does not meet the completion condition, we define the profit obtained from subtask $t_{j,k}$ as the potential profit $V(A_p(t_{j,k}))$. In this case, we get

$$V(A_p(t_{j,k})) = b_{j,k} \frac{|s(A_p(t_{j,k}))|}{|sk_{j,k}|} - C(A_p(t_{j,k})), \tag{1}$$

where $|s(A_p(t_{j,k}))|$ indicates the number of required skills provided by the matched workers in $A_p(t_{j,k})$, and we can get $|s(A_p(t_{j,k}))| = |\left(\sum_{\langle t_{j,k}, w_i \rangle \in A_p(t_{j,k})} sk_i\right) capsk_{j,k}|$. In addition, $C(A_p(t_{j,k}))$ is the sum of the travel cost of workers in $A_p(t_{j,k})$ to the location $l_{j,k}$ of subtask $t_{j,k}$, and $C(A_p(t_{j,k})) = \sum_{\langle t_{j,k}, w_i \rangle \in A_p(t_{j,k})} c_{i,j,k}$, where $c_{i,j,k}$ is the travel expenses of worker w_i to the location of subtask $t_{j,k}$.

Since $c_{i,j,k}$ is related to the distance between worker w_i and subtask $t_{j,k}$, we can obtain $c_{i,j,k} = c \times dist(l_i, l_{j,k})$. $dist(l_i, l_{j,k})$ is the Euclidean distance between worker w_i and subtask $t_{j,k}$. Following the work in [12], we use the Euclidean distance to represent the distance $dist(l_i, l_{j,k})$ between worker w_i and subtask $t_{j,k}$. Without loss of generality, it can also be easily adjusted into the shortest path distance in road networks. Additionally, c is the cost of moving unit distance. For vehicles, c is the gas fee per mile [12]. For electric cars, c is the electricity consumption per mile. If subtask $t_{j,k}$ meets the completion condition (i.e. the required skills of $t_{j,k}$ can be fully covered by the skills of the matched workers in $A_p(t_{j,k})$), we can get $|s(A_p(t_{j,k}))| = |sk_{j,k}|$ and define the profit obtained from subtask $t_{j,k}$ as $V(A_p(t_{j,k})) = b_{j,k} - C(A_p(t_{j,k}))$.

Definition 4. (Multi-stage Complex Task Assignment Problem). In a timestamp interval $[0, p]$, a batch of workers W_p and tasks T_p appear on the crowdsourcing platform. The problem of multi-stage complex task assignment in spatial crowdsourcing is to obtain an assignment A_p, such that:

- the matching pairs $\langle t_{j,k}, w_i \rangle \in A_p$ are valid matching pairs;
- the total profit $V(A_p)$ is maximized, where

$$\begin{aligned} V(A_p) &= \sum_{t_{j,k} \in CT_p} V(A_p(t_{j,k})) \\ &= \sum_{t_{j,k} \in CT_p} b_{j,k} - C(A_p(t_{j,k})), \end{aligned} \tag{2}$$

where CT_p is a set of subtasks that meet the completion condition.

It is worth noting that the total profit $V(A_p)$ is affected by the number of completed subtasks and the profit obtained from each subtask. Accordingly, first, workers tend to choose subtasks that meet the completion condition, and second, if the number of assigned workers exceeds the skill requirements of subtask $t_{j,k}$, we keep a set of workers who can complete the subtask and obtain the maximum profit.

Since $V(A_p)$ is the total profit after deducting the travel expenses of workers, we can allocate $V(A_p)$ based on the service hours and skills provided by the workers, which will be studied in our future work.

Theorem 1. *The problem of Multi-stage Complex Task Assignment is NP-hard.*

Proof. We prove this theorem by a reduction from an existing NP-hard problem, namely the K-set packing problem [34], defined as follows: given a set of elements $Q = \{q_1, q_2, \cdots, q_{|e|}\}$, a collection of subsets $P = \{P_1, P_2, \cdots, P_{|s|}\}$, and a number K, where $P_j \subseteq Q$ and each P_j is relevant to a weight value $w(P_j)$; the K-set packing problem is to choose a collection of subsets $P^* \subseteq P$ to maximize $\sum_{P_j \in P^*} w(P_j)$, where any two subsets $P_j, P_k \in P^*$ are disjoint and the number of elements in any subset $P_j \in P^*$ is at most K.

For a given K-set packing problem, we can transform it to an MSCTA instance within polynomial time. We configure that each task only consists of one stage (i.e., $t_j = t_{j,1}$) and needs at most K skills to meet the completion condition (i.e., $|sk_j| \leqslant K$). Each worker w_i corresponds to an element $q_i \in Q$, each subset $P_j \in P$ is associated to a set of workers $W_j = \{w_i | \langle t_j, w_i \rangle \in A_p(t_j)\}$ assigned to task t_j. We configure that each task t_j can be accomplished by its corresponding set of workers W_j. The profit obtained by the workers in W_j equals to $w(P_j)$ (i.e., $V(A_p(t_j)) = w(P_j)$). Since $|sk_j| \leqslant K$, the number of workers in each set W_j is at most K. For this MSCTA instance, our purpose is to select a set of tasks to complete such that the total profit $V(A_p)$ is maximized, which is the same to maximize $\sum_{P_j \in P^*} w(P_i)$ in the K-set packing instance. Under this polynomial-time mapping method, we can obtain that the K-set packing instance can be solved if and only if the transformed MSCTA problem can be solved.

Therefore, we reduce the K-set packing problem to the MSCTA problem. Since the K-set packing problem is NP-hard [34], the MSCTA problem is also NP-hard.

4. Batch-based framework

In this section, we present a framework to address the MSCTA problem.

In Algorithm 1, we design a batch-based framework that iteratively assigns tasks to workers in multiple batches. Compared with the online task assignment mode that needs to assign tasks to workers immediately, the batch-based framework can obtain a better assignment result and is suitable for dynamic scenarios [1]. Specifically, at each timestamp $p \in P$, the crowdsourcing platform collects a batch of tasks T_p and workers W_p (lines 2–3). T_p includes the tasks that are not assigned enough skilled workers before timestamp p and the newly arrived tasks before timestamp p. W_p consists of the following three categories: the workers who are not assigned any tasks before timestamp p, the workers that have completed their tasks and reconnected to the platform before timestamp p, and the workers who are newly connected to the platform before timestamp p.

Algorithm 1: Batch-based Framework

Input: A time interval P
Output: Task-and-worker assignment results within P
1: for each timestamp $p \in P$
2: Collect a set of tasks T_p
3: Collect a set of workers W_p
4: Apply the greedy algorithm or the game algorithm to obtain a good assignment A_p
5: **end for**

Since the MSCTA problem is NP-hard, we propose a greedy algorithm and a game algorithm to obtain a good assignment A_p (line 4). By applying the greedy algorithm, we can get a local optimal assignment A_p based on the SAT mode [22], suffering the limitation of workers cannot choose tasks independently. Differently, in the game algorithm, workers can choose tasks according to their preferences and reach stability.

Before describing the greedy algorithm and the game algorithm, we introduce two operations that are often invoked by them. The first operation is called valid candidate judgments (*VCJ*), applied to obtain a set of valid candidate workers for each subtask. The second operation, called *filtering*, is applied to extract a set of executable subtasks *ET*.

For the operation of *VCJ* mentioned in Algorithm 2, to obtain a set of valid candidate workers for each subtask (lines 3–9), we set some judgment conditions, including the constraints of the maximal working area, deadline, skills, and budget. For example, to obtain a set of valid candidate workers $w_i \in VC_{j,k}$ for subtask $t_{j,k}$, worker w_i and subtask $t_{j,k}$ must satisfy the following *judgement conditions*:

Algorithm 2: Valid Candidate Judgments (VCJ)

Input: A batch of tasks T_p and workers W_p
Output: A set of valid candidate workers *VC*
1: $VC \leftarrow \varnothing$
2: **for** each subtask $t_{j,k} \in T_p$ **do**
3: **for** each worker $w_i \in W_p$ **do**
4: **if** w_i has not been assigned **then**
5: **if** $\langle t_{j,k}, w_i \rangle$ satisfies *judgement conditions* **then**
6: $VC_{j,k} \leftarrow VC_{j,k} \cup \{w_i\}$
7: **end if**
8: **end if**
9: **end for**
10: $flag_{j,k} \rightarrow 1$
11: Update *VC*
12: **end for**
13: Return *VC*

- Maximal working area. The distance between worker w_i and subtask $t_{j,k}$ must be less than d_i,

$$dist(l_i, l_{j,k}) = \sqrt{\left(l_{j,k}(x) - l_i(x)\right)^2 + \left(l_{j,k}(y) - l_i(y)\right)^2} \leqslant d_i.$$

- Budget constraint. The budget $b_{j,k}$ should be able to compensate worker w_i for travel expenses from l_i to $l_{j,k}$, and we get $dist(l_i, l_{j,k}) \times c \leqslant b_{j,k}$.
- Deadline constraint. Each worker w_i must reach the location $l_{j,k}$ before the deadline $(a_j + wt_{j,k})$, and we obtain $\frac{1}{v_i} \times dist(l_i, l_{j,k}) \leqslant (a_j + wt_{j,k} - p)$.
- Skills constraint. Each worker w_i must provide some skills required by subtask $t_{j,k}$, and we have $sk_i capsk_{j,k} \neq \varnothing$.

Additionally, we set the flag $flag_{j,k}$ of subtask $t_{j,k}$ to 1, which means that subtask $t_{j,k}$ does not reach the completion condition (line 10).

Algorithm 3: Filtering

Input: A set of tasks T_p and *VC*
Output: A set of executable tasks *ET*
1: Initialize *ET*
2: **for** each subtask $t_{j,k} \in T_p$ **do**
3: **if** $flag_{j,k} = 1, D_{j,k} = \varnothing$ and $VC_{j,k} \neq \varnothing$ **then**
4: $ET \leftarrow ET \cup \{t_{j,k}\}$
5: **end if**
6: **end for**
7: **Return** *ET*

The *filtering* operation in Algorithm 3 is used to obtain a set of executable subtasks (ET). Specifically, if subtask $t_{j,k}$ meets the following conditions: $flag_{j,k} = 1, D_{j,k} = \varnothing$, and $VC_{j,k} \neq \varnothing$, then we add subtask $t_{j,k}$ to ET (lines 2–6).

Please note that our batch-based framework is also suitable for such a case, i.e. a whole task may become meaningless due to some subtasks that cannot be completed. Specifically, for some subtasks that cannot meet the completion conditions in the current batch, we will add them to the next batch and give priority to them.

5. The greedy approach

In this section, we introduce a greedy algorithm that assigns the most profitable workers to subtasks.

5.1. The definition of profit increase

Before introducing the greedy algorithm, we give a definition of profit increase $\Delta V(t_{j,k}, w_i)$ by using the case of assigning a valid candidate worker w_i to subtask $t_{j,k}$. For subtask $t_{j,k}, A_p(t_{j,k})$ is an assignment result including the matching pair $\langle t_{j,k}, w_i \rangle$, while $A_p(t_{j,k}) \setminus \{\langle t_{j,k}, w_i \rangle\}$ indicates that worker w_i is not assigned to subtask $t_{j,k}$. Therefore, the profit increase can be calculated by

$$\Delta V(t_{j,k}, w_i) = V(A_p(t_{j,k})) - V(A_p(t_{j,k}) \setminus \langle t_{j,k}, w_i \rangle). \tag{3}$$

5.2. The Greedy Algorithm

Based on the definition of profit increase $\Delta V(t_{j,k}, w_i)$, we design a greedy algorithm in Algorithm 4 to solve the MSCTA problem.

Algorithm 4: The Greedy Algorithm

Input: A batch of tasks T_p and workers W_p
Output: A assignment result A_p
1: $A_p \leftarrow \varnothing$
2: $VC \leftarrow$ Use *VCJ* designed in Algorithm 2
3: $ET \leftarrow$ Use *filtering* designed in Algorithm 3
4: **While** A_p is not stable **do**
5: $I \leftarrow \varnothing$
6: **for** each subtask $t_{j,k} \in ET$ **do**
7: Select worker w_i with assignment strategies
8: $I \leftarrow I \cup \{\langle t_{j,k}, w_i \rangle\}$
9: end for
10: **For** each pair $\langle t_{j,k}, w_i \rangle \in I$ **do**
11: **if** worker w_i is assigned to multiple subtasks **then**
12: Assign w_i to the subtask with the largest profit increase and update I
13: **end if**
14: **end for**
15: $A_p \leftarrow A_p \cup I$
16: $VC \leftarrow VC \setminus \{w_i | \langle t_{j,k}, w_i \rangle \in I\}$
17: **for** each subtask $t_{j,k} \in ET$ **then**
18: **if** $t_{j,k}$ meets the completion condition **then**
19: $T_p \leftarrow T_p \setminus \{t_{j,k}\}$
20: $D_{j,a} \leftarrow D_{j,a} \setminus \{t_{j,k}\}$, where $(k < a \leqslant l)$
21: $flag_{j,k} \rightarrow 0$
22: **end if**
23: **end for**
24: Apply the *filtering* algorithm to update ET
25: end while
26: Return A_p

We first initialize the result of assignment A_p to an empty set (line 1). Then, according to Definition 3, we apply the *VCJ* algorithm to obtain a set of valid candidate workers $VC_{j,k}$ for each subtask $t_{j,k}$ (line 2). In addition, considering the dependency constraints among subtasks, we apply a *filtering* algorithm to obtain a set of executable subtasks *ET* (line 3). Finally, based on *VC* and *ET*, we design a WHILE loop to iteratively match the best worker to each executable subtask (lines 4–25). Specifically, for each iteration in the WHILE loop, we first initialize a temporary set I to an empty set to store the newly matched task-and-worker pairs (line 5), and then we select the best worker for each subtask in *ET* according to the assignment strategies (lines 6–9). The WHILE loop terminates when A_p reaches stability (line 4), which means that the current assignment result A_p has not changed compared with the result of the previous round.

Our goal is to maximize the total profit $V(A_p)$, which is not only affected by the number of completed subtasks, but also by the profit obtained from each subtask. To achieve this goal, we can choose the *best* worker for each subtask according to the following assignment strategies:

- The selected worker w_i must guarantee that the profit increase $\Delta V(t_{j,k}, w_i) > 0$;
- Workers who can make subtask $t_{j,k}$ meet the completion conditions mentioned in Section 3 will be given priority. For example, subtask $t_{j,k}$ has two valid candidate workers w_i and w_j that can be selected and can obtain two assignment results $A_p(t_{j,k}) \cup \{\langle t_{j,k}, w_i\rangle\}$ and $A_p(t_{j,k}) \cup \{\langle t_{j,k}, w_j\rangle\}$, respectively. If subtask $t_{j,k}$ can be completed by the assignment $A_p(t_{j,k}) \cup \{\langle t_{j,k}, w_i\rangle\}$ but not by $A_p(t_{j,k}) \cup \{\langle t_{j,k}, w_j\rangle\}$, then worker w_i will be assigned to subtask $t_{j,k}$;
- Under the premise of satisfying the above assignment strategies, workers who can provide the higher profit increase will be selected.

If worker w_i is the best choice for several subtasks, then we compare the profit increase of different subtasks, and assign worker w_i to the subtask with the largest profit increase and update I (lines 10–14). Additionally, we update A_p with I (line 15) and remove these assigned workers from *VC* (line 16). If any subtask in *ET* meets the completion condition, we update T_p and $D_{j,a}(k < a \leqslant l)$, and set the flag $flag_{j,k}$ of subtask $t_{j,k}$ to 0, which indicates that subtask $t_{j,k}$ reaches the completion condition mentioned in Definition 3 (lines 17–23). Finally, we apply the *filtering* algorithm to obtain a new set of executable tasks *ET* (line 24). After several iterations, we can obtain a stable assignment result A_p, i.e. the currently executable tasks cannot match more workers (line 26).

5.3. Analysis of the Greedy Algorithm

In this subsection, we first analyze the time complexity of the greedy algorithm, and then discuss the result obtained by the algorithm.

Theorem 2. *The time complexity of the greedy algorithm is* $O(\max(nml, mn^2, m^2n))$.

Proof. In Algorithm 4, we assume that each worker w_i $(1 \leqslant i \leqslant n)$ is a valid candidate worker for each subtask $t_{j,k}$ $(1 \leqslant j \leqslant m, 1 \leqslant k \leqslant l)$. The time required to obtain a set of valid candidate workers for each subtask is $O(nml)$ (line 2). The time required to obtain a set of executable tasks is $O(ml)$ (line 3). In the WHILE loop, at least one worker is assigned to a subtask in each iteration, so there are at most n iterations. In each iteration, the maximum number of executable subtasks in *ET* is m and it takes $O(mn)$ time to assign a worker to each executable subtask (lines 6–9). To prevent workers from being redistributed, $O(m^2)$ time is required to compare all of the matching pairs in I (lines 10–14). Moreover, $O(m)$ time is required to check whether any subtasks in *ET* satisfy the completion condition (lines 17–22), and then, $O(ml)$ time is necessary to update *ET* (line 23). Therefore, the maximum time complexity of the greedy algorithm is $O(\max(nml, mn^2, m^2n))$.

To analyze the result of the algorithm, we first prove the following theorem.

Theorem 3. *The total profit* $V(A_p)$ *obtained by the greedy algorithm is monotonic and submodular.*

Proof. We first prove the monotonicity of $V(A_p)$. In the greedy algorithm, we select the best worker for each subtask in turn in each round. At timestamp t^h, the assignment result is A_p^h, and the total profit is $V\left(A_p^h\right)$. At timestamp t^{h+1}, we assume that worker w_i is assigned to subtask $t_{j,k}$, so the assignment result is $A_p^{h+1} = A_p^h \cup \langle t_{j,k}, w_i\rangle$ and the total profit is $V\left(A_p^{h+1}\right)$. According to Eq. (2), we have $V\left(A_p^{h+1}\right) - V\left(A_p^h\right) = V\left(A_p^h(t_{j,k}) \cup \langle t_{j,k}, w_i\rangle\right) - V\left(A_p^h(t_{j,k})\right)$. Based on Eq. (3), we can obtain the profit increase

as $\Delta V(t_{j,k}, w_i) = V\left(A_p^h(t_{j,k}) \cup \langle t_{j,k}, w_i \rangle\right) - V\left(A_p^h(t_{j,k})\right)$. According to assignment strategies in the greedy algorithm, we obtain $\Delta V(t_{j,k}, w_i) > 0$. Therefore, the total profit $V(A_p)$ obtained by the greedy algorithm is monotonic.

According to [35], to prove the submodularity of $V(A_p)$, it requires to prove that $V(A_p \cup \langle t_{j,k}, w_i \rangle) - V(A_p) \geqslant V\left(A'_p \cup \langle t_{j,k}, w_i \rangle\right) - V\left(A'_p\right)$, where $A_p \subseteq A'_p \subseteq \widehat{A_p}$, and a matching pair $\langle t_{j,k}, w_i \rangle$ is included in $\widehat{A_p}$ but not in A'_p. Since $A_p \subseteq A'_p$, we can get $A_p(t_{j,k}) \subseteq A'_p(t_{j,k})$. According to Eq. (1) and Eq. (2), we have

$$\begin{aligned} V(A_p \cup \langle t_{j,k}, w_i \rangle) - V(A_p) &= V(A_p(t_{j,k}) \cup \langle t_{j,k}, w_i \rangle) - V(A_p(t_{j,k})) \\ &= b_{j,k} \frac{|s(A_p(t_{j,k}) \cup \langle t_{j,k}, w_i \rangle)| - |s(A_p(t_{j,k}))|}{|sk_{j,k}|} - c_{i,j,k} \\ &\geqslant b_{j,k} \frac{|s(A'_p(t_{j,k}) \cup \langle t_{j,k}, w_i \rangle)| - |s(A'_p(t_{j,k}))|}{|sk_{j,k}|} - c_{i,j,k} \\ &= V\left(A'_p(t_{j,k}) \cup \langle t_{j,k}, w_i \rangle\right) - V\left(A'_p(t_{j,k})\right) \\ &= V\left(A'_p \cup \langle t_{j,k}, w_i \rangle\right) - V\left(A'_p\right). \end{aligned} \tag{4}$$

Therefore, the total profit $V(A_p)$ obtained by the greedy algorithm is submodular.

Based on the above analysis, we have proved that the total profit $V(A_p)$ obtained by the greedy algorithm is nonnegative, monotonic and submodular, according to [35], the greedy algorithm can achieve a result with guaranteed approximate bounds, that is, $(1 - \frac{1}{e}) \cdot V\left(\widetilde{A_p}\right)$, where $\widetilde{A_p}$ is the global optimal assignment result.

6. The game approach

In real-world crowdsourcing applications, workers can log on to the platform freely and select tasks according to their preferences. Although the greedy algorithm can obtain a local optimal assignment A_p based on the SAT mode [22], it faces the limitation that workers cannot choose tasks independently. Inspired by this fact, we introduce a game algorithm, in which each worker can repeatedly adjust his/her selection according to the strategies of others until reaching stability.

6.1. The game theory

Before proposing our game algorithm, we introduce some information about game theory [36–39] in this subsection.

A strategic game consists of a set of players, and each player has a set of strategies. In addition, for each player, a utility function is used to measure the utility value of each strategy of the player. Moreover, an essential feature of a strategic game is that each player's utility depends on the list of all the other players' strategies. There are n players in a strategic game $G = (S, u)$, where S denotes the Cartesian product of user strategies, that is, $S = S_1 \times S_2 \times \cdots \times S_n$, and S_i is the strategy set of player i. $u = (u_1, u_2, \cdots, u_n)$ represents the *utility profile* composed of all users. For player i $(1 \leqslant i \leqslant n)$, in each round of the game, he/she makes a strategy $s_i \in S_i$ to maximize his/her utility u_i depending on the strategies s_{-i} of other players.

The purpose of the game $G = (S, u)$ is to find a Nash equilibrium $s^* \in S$, where no one wants to change his/her strategy unilaterally, because each player i $(1 \leqslant i \leqslant n)$ can meet the condition $u_i(s_i^*, s_{-i}^*) \geqslant u_i(s_i, s_{-i}^*)$, where $s^* = (s_i^*, s_{-i}^*)$. That is to say, a Nash equilibrium of a strategic game can be regarded as a strategy profile, and its attribute is that no player can increase her/is utility by choosing a different action, given the other players' actions.

For example, in a strategy game, there are two players A and B, and they have two strategies, L and M. We can get four strategy profiles $(L, L), (L, M), (M, L)$, and (M, M), correspondingly there are four utility combinations $(2, 2), (0, 3), (3, 0)$, and $(1, 1)$. The first action in each strategy profile is player A's strategy and the first number in each utility combination is player A's payoff to the corresponding strategy profile. Therefore, if player A chooses the action L and player B chooses the action M, then player A's utility is 0 and player B's utility is 3. To find a Nash equilibrium, we can examine each strategy profile in turn. For the strategy profile (L, L), by choosing M rather than L, player A obtains a utility of 3 rather than 2, given player B's strategy. Thus (L, L) is not a Nash equilibrium. Additionally, Player B also can increase her utility by choosing strategy M rather than L. In the same way, neither (L, M) nor (M, L) is a Nash equilibrium. However, for the strategy profile (M, M), neither player can increase her/is utility by choosing a strategy different from the current one. Therefore, the strategy profile (M, M) is a Nash equilibrium.

6.2. The Game Algorithm

In the context of game theory, the MSCTA problem can be modeled as a game, namely the MSCTA game. The collected workers are players in the MSCTA game, and the strategic space is the set of tasks T_p. In addition, the purpose of the MSCTA game is to maximize the total profit $V(A_p)$.

Algorithm 5: The MSCTA Game

Input: A set of tasks T_p and workers W_p
Output: An assignment result A_p
1: Apply the Greedy Algorithm to get a initial result A_p
2: $VC \leftarrow$ Use *VCJ* designed in Algorithm 2
3: $ET \leftarrow$ Use *filtering* designed in Algorithm 3
4: **While** ET is not stable **then**
5: **While** Nash equilibrium is not reached **then**
6: **for** worker $w_i \in W_p$ **do**
7: Select subtask $t_{j,k} \in ET$ by selection strategies
8: Update $A_p(t_{j,k})$
9: **end for**
10: **end while**
11: **for** each subtask $t_{j,k} \in ET$ **do**
12: **if** $t_{j,k}$ meets the completion condition **then**
13: $flag_{j,k} \rightarrow 0$
14: $T_p \leftarrow T_p \setminus \{t_{j,k}\}$
15: $D_{j,a} \leftarrow D_{j,a} \setminus \{t_{j,k}\}$, where $(k < a \leqslant l)$
16: $VC \leftarrow VC \setminus \{w_i | \langle t_{j,k}, w_i \rangle \in A_p(t_{j,k})\}$
17: $W_p \leftarrow W_p \setminus \{w_i | \langle t_{j,k}, w_i \rangle \in A_p(t_{j,k})\}$
18: **end if**
19: **end for**
20: Apply the *filtering* algorithm to update ET
21: **end while**
22: Return A_p;

By defining the utility function u_i of each worker w_i $(1 \leqslant i \leqslant n)$ as the profit increase $\Delta V(t_{j,k}, w_i)$ in Eq. (3), we obtain

$$\begin{aligned} u_i(s_i, s_{-i}) &= \Delta V(t_{j,k}, w_i) \\ &= V(A_p(t_{j,k})) - V(A_p(t_{j,k}) \setminus \langle t_{j,k}, w_i \rangle), \end{aligned} \tag{5}$$

where s_i represents the strategy of worker w_i and s_{-i} is the strategy vector of all workers except w_i.

In Algorithm 5, we propose a game algorithm to solve the MSCTA problem. We first initialize the assignment A_p (line 1). Then, according to Definition 3, we apply the *VCJ* algorithm to obtain a set of valid candidate workers $VC_{j,k}$ for each subtask $t_{j,k} \in T_p$ (line 2). In addition, since subtasks belonging to the same task are dependent, we apply the *filtering* algorithm to get a set of executable tasks ET (line 3). Taking ET as the current strategy space, based on the strategies of other workers, each worker $w_i \in W_p$ iteratively optimizes his/her strategy to maximize his/her own utility u_i until the inner WHILE loop reaches a Nash equilibrium (lines 5–10). Additionally, each worker w_i selects the best subtask $t_{j,k} \in ET$ according to the following selection strategies (line 7):

- the selected subtask $t_{j,k}$ must guarantee that the utility of worker w_i is greater than 0, $u_i = \Delta V(t_{j,k}, w_i) > 0$;
- worker w_i preferentially selects the subtasks that meet the completion condition mentioned in Section 3. For example, worker w_i has two valid candidate subtasks $(t_{j,k}, t_{l,h})$ to select and can obtain two assignment results $A_p(t_{j,k}) \cup \{\langle t_{j,k}, w_i \rangle\}$ and $A_p(t_{l,h}) \cup \{\langle t_{l,h}, w_j \rangle\}$, respectively. If subtask $t_{j,k}$ can be completed by the assignment $A_p(t_{j,k}) \cup \{\langle t_{j,k}, w_i \rangle\}$, while subtask $t_{l,h}$ cannot be completed by the assignment $A_p(t_{l,h}) \cup \{\langle t_{l,h}, w_i \rangle\}$, then worker w_i will select subtask $t_{j,k}$;
- under the premise of satisfying the above selection strategies, worker w_i will select the subtask that can bring higher utility.After the internal WHILE loop is completed, the algorithm checks whether any subtasks in ET have reached the completion condition mentioned in Definition 3 and makes the following updates: update the identifier $flag_{j,k}$ indicating that $t_{j,k}$ has reached the completion condition to 0, remove $t_{j,k}$ from T_p and $D_{j,a}(k < a \leqslant l)$ respectively, remove w_i from W_p and VC respectively (lines 11–19). Finally, the algorithm applies the *filtering* algorithm to obtain a new set of executable subtasks ET (line 20) and continue to execute the external WHILE loop until ET reaches stability.

6.3. Analysis of the Game Algorithm

In this subsection, we analyze the characteristics of the MSCTA game from the following aspects: the stability, the convergence rate, and the result quality.

The stability. According to the introduction to game theory in Section 4, the internal WHILE loop can be considered as a game $g = (s, u)$ where workers included in W_p are the players and the set of executable subtasks ET is the strategic space of the players. To analyze the stability of the MSCTA game, we first discuss whether the internal WHILE loop can reach a Nash equilibrium, and then, we analyze whether the external WHILE loop can reach stability.

Before proving that the internal WHILE loop can reach a Nash equilibrium, we introduce the definition and properties of potential games [40]:

- A game $G = (s, u)$ is a potential game if and only if there exists a potential function p such that $u_i(s_i, s_{-i}) - u_i(s'_i, s_{-i}) = p(s_i, s_{-i}) - p(s'_i, s_{-i})$, where u_i is the utility of player w_i, s_i and s'_i are two different strategies of play w_i, and s_{-i} denotes the vector of all workers' strategies except for that of player w_i.
- For a potential game $G = (s, u)$, if the strategy set of workers is a finite set, then it always converges to a pure Nash equilibrium.

Inspired by the definition of potential game and the properties of potential game, we have the following theorem.

Theorem 4. *The internal WHILE loop (lines 5–10 in Algorithm 5) can reach a Nash equilibrium.*

Proof. We first prove that the internal WHILE loop is a potential game. We take the total profit function $V(A_p)$ mentioned in Eq. (4) as the potential function p. In addition, s_i and s'_i indicate that worker w_i chooses subtask $t_{j,k}$ and subtask $t'_{j,k}$, respectively. Moreover, according to Algorithm 5, $t_{j,k}$ and $t'_{j,k}$ are executable tasks in ET. Thus, we obtain

$$\begin{aligned}
&p(s_i, s_{-i}) - p(s'_i, s_{-i}) \\
&= \left(V(A_p(t_{j,k})) + V\left(A_p\left(t'_{j,k}\right) \setminus \langle t'_{j,k}, w_i\rangle\right) + \sum_{t_{x,y} \in CT'_p} V(A_p(t_{x,y})) \right) \\
&\quad - \left(V\left(A_p\left(t'_{j,k}\right)\right) + V(A_p(t_{j,k}) \setminus \langle t_{j,k}, w_i\rangle) + \sum_{t_{x,y} \in CT'_p} V(A_p(t_{x,y})) \right) \\
&= \left(V(A_p(t_{j,k})) - V(A_p(t_{j,k}) \setminus \langle t_{j,k}, w_i\rangle)\right) \\
&\quad - \left(V\left(A_p\left(t'_{j,k}\right)\right) - V\left(A_p\left(t'_{j,k}\right) \setminus \langle t'_{j,k}, w_i\rangle\right)\right) \\
&= u_i(s_i, s_{-i}) - u_i(s'_i, s_{-i}),
\end{aligned} \tag{6}$$

where, $CT'_p = CT_p \setminus \left\{t_{j,k}, t'_{j,k}\right\}$. Consequently, the internal WHILE loop is a potential game. According to the properties of a potential game, since the strategic space of each worker in the internal WHILE loop is limited, it can be concluded that the internal WHILE loop can find a Nash equilibrium.

We now analyze the stability of the external WHILE loop. In each round of the external WHILE loop, if any subtasks in ET reach the completion condition mentioned in Section 3 after the internal WHILE loop reaches a Nash equilibrium, then we remove these subtasks from ET and apply the *filtering* algorithm to get a new set of executable tasks ET. If there are sufficient workers, ET eventually becomes an empty set, that is, all subtasks meet the completion condition mentioned in Definition 3. If there are not enough workers, ET will not be updated continuously because no subtasks contained in ET meet the completion condition.

From the above analysis of the internal WHILE loop and the external WHILE loop, it can be concluded that the MSCTA game can reach stability.

The convergence rate. To discuss the convergence rate of the MSCTA game, it is necessary to analyze the number of rounds required for the external WHILE loop and the internal WHILE loop to reach stability and to calculate the time complexity of each round.

Since the internal WHILE loop can be regarded as a potential game, to estimate the upper limit of the number of rounds required for the internal WHILE loop to reach a Nash equilibrium, we discuss a special case where the utility function value of each worker is an integer. We assume that $p_z(s) = z \times p(s)$, where z is a multiplicative factor that makes $p_z(s) \in Z$.

Lemma 1. For the internal WHILE loop, the number of rounds necessary to achieve a Nash equilibrium must be less than $p_z(s^*)$, where s^* is the optimal joint strategy of the workers that can select the tasks in the internal WHILE loop.

Proof. A worker w_i changes his selection from current strategy s_i to a better strategy s'_i, and the value of $p_z(s)$ should increase by at least 1 for the following reasons: 1) $p_z(s) \in Z$; and 2) $p(s_i, s_{-i}) - p(s'_i, s_{-i}) = u_i(s_i, s_{-i}) - u_i(s'_i, s_{-i}) > 0$. Therefore, the value of the potential function $p_z(s)$ increases until the optimal joint strategy s^* is reached, where no one changes his/her current strategy unilaterally. Consequently, the number of rounds required for the internal WHILE loop to find a Nash equilibrium must be less than $p_z(s^*)$. The proof of the lemma has been completed.

Next, we compute the value range of $p_z(s^*)$. Since the strategic space in the internal WHILE loop is ET and $p_z(s^*) \in Z$, according to Eq. (4), we can get $p_z(s^*) \leqslant \sum_{t_{j,k} \in ET} \lceil b_{j,k} \rceil$.

For the external WHILE loop, the number of rounds required to reach stability depends on the number of updates of ET. When the internal WHILE loop reaches equilibrium, if only one subtask meets the completion condition, then the maximum number of rounds required for the external WHILE loop to reach a Nash equilibrium is ml.

Since the number of rounds required for the external WHILE loop and the internal WHILE loop to reach stability has been calculated, we can obtain that the maximum time complexity of the MSCTA game is $O\left(\max\left(m^2l^2n, m^2nlp_z(s^*)\right)\right)$. Specifically, in Algorithm 5, to obtain a set of valid candidate workers for each subtask, the required time is $O(nml)$ (line 2). To get a set of executable tasks, the required time is $O(ml)$ (line 3). In each round of the internal WHILE loop, there are at most m executable subtasks in ET, so that $O(mn)$ time is required for the workers to select the best subtasks (lines 6–9). Moreover, since the maximum number of rounds required for the internal WHILE loop to reach an equilibrium is $p_z(s^*)$, we obtain that the time complexity of the internal WHILE loop is $O(p_z(s^*)mn)$ (lines 5–10). Then, $O(m)$ time is required to update $flag_{j,k}, T_p, D_{j,a}(k < a \leqslant l), W_p$ and VC (lines 11–19). In addition, the maximum number of rounds required for the external WHILE loop to reach stability is ml. In summary, the maximum time complexity of the MSCTA game is $O\left(\max\left(m^2l^2n, m^2nlp_z(s^*)\right)\right)$.

The result quality. To evaluate the quality of the result achieved by the MSCTA game, we can apply the following three measures [40]:

- *social optimum (OPT).* OPT is the global maximum value of the objective function.
- *price of stability (PoS).* PoS is the ratio of the maximum value obtained in all Nash equilibrium solutions to the global maximum value, which indicates the upper bound of the ratio of the objective function value obtained by Nash equilibrium to the global maximum value.
- *price of anarchy (PoA).* PoA is the ratio of the minimum value obtained among all Nash equilibrium solutions to the global maximum value, which is the lower bound of the ratio of the objective function value obtained by Nash equilibrium to the global maximum value.

Theorem 5. *In the MSCTA game, the maximum value of PoS is 1 and the minimum value of PoA is* $\frac{V_{min}(t) \cdot num(init)}{V\left(\widetilde{A_p}\right)}$, *where* $V_{min}(t)$ *is the minimum profit obtained from any subtask that meets the completion condition in the initial assignment* $A_{p,init}$ *and* $num(init)$ *is the number of subtasks that satisfy the completion condition in* $A_{p,init}$.

Proof. We define the global optimal task-and-worker assignment as $\widetilde{A_p}$, the assignment of the largest objective function value in all Nash equilibrium solutions is represented as A_p^*. In addition, the total profit can be described as $V(A_p) = p(A_p)$. Therefore, we can get $OPT = V\left(\widetilde{A_p}\right) \geqslant V\left(A_p^*\right)$. As a result, $PoS = \frac{V\left(A_p^*\right)}{OPT} \leq 1$.

Since the internal WHILE loop is a potential game, we obtain that the profit increase of each worker is equal to the increase in the total profit, and the total profit will continue to increase until a Nash equilibrium is reached. In addition, the MSCTA game is based on the result of the greedy algorithm, and $A_{p,init}$ is the initial assignment of the game algorithm, $V_{min}(t)$ is the minimum profit obtained from any subtask that meets the completion condition in the initial assignment $A_{p,init}$ and $num(init)$ is the number of subtasks that satisfy the completion condition in $A_{p,init}$. Therefore,

$$PoA = \frac{V(A_{p,init})}{OPT} \geqslant \frac{V_{min}(t) \cdot num(init)}{V\left(\widetilde{A_p}\right)}. \tag{7}$$

The proof of the theorem has been completed.

7. Experimental study

In this part, we first describe the data sets used in our experiments, then introduce the approaches applied to solve the MSCTA problem, and finally conduct experiments to measure the effect of different parameters on the results.

Table 4
Parameter Information of Real Data Set from [3].

Parameters	Values
the number of subtasks, ml	5 k, 6 k, **8 k**, 10 k
the number of workers, n	3 k, **5 k**, 7 k, 9 k

Table 5
Parameter Information of Real Data Set from [41].

Parameters	Values
the budget for subtasks, $[b^-, b^+]$	[1, 5], [6, 10], **[11, 15]**, [16, 20]
the start time range, $[a^-, a^+]$	[0, 5], [0, 10], **[0, 15]**, [0, 20]

7.1. Data Sets

We test our proposed algorithm by applying synthetic and real data sets, and refer to [12] to set all parameters.

To test our algorithms, we use two real data sets from [3,41], and the default values of the parameters are in bold font in Table 4 and Table 5, respectively. For the data set from [3], we extract part of the records of Chengdu, China, on November 11, 2016, including 10000 tasks and 9000 workers. For the Meetup dataset from [41], we extract records of Chicago in November 2017, containing 1000 workers and 1000 tasks.

For both real data sets, we use the positions of users and events to initialize the locations of workers and tasks in the MSCTA problem, which are linearly mapped into a $[0, 1]^2$ space. We set every ten events to form a multi-stage complex task and establish dependencies according to the start processing time of each task in both real data sets; the size of the skill universe $|s|$ is set to 70. The waiting time wt of each event is randomly selected in $[1, 15]$ and ensures that the waiting time of a subtask is longer than the subtasks it depends on. The velocity v of each worker is randomly distributed in the range of $[0.001, 0.01]$; the maximum moving distance d of each worker is set as 0.2 and the cost of moving unit distance c is 10. For the data set from [41], we test the impact of budget range $[b^-, b^+]$ and start time range $[a^-, a^+]$ on the assignment results, varing from $[1, 5]$ to $[16, 20]$ and $[0, 5]$ to $[0, 20]$, respectively. The skills of workers and tasks are set according to the tags of users and events in the data set. For the data set from [3], we test the impact of the number of workers and the number of tasks on the assignment results, from 3 k to 9 k and 5 k to 10 k, respectively. Additionally, the number of skills required for each subtask is randomly selected within $[1, 3]$ and each worker has one skill.

For the synthetic data set, the values of the parameters are shown in Table 6. Specifically, the locations of workers and tasks are randomly generated in a 2D data space $[0, 1]^2$. For the tasks, the total number of the subtasks ml varies from 300 to 1200 at an increment of 300; the number of stages l contained in each multi-stage complex task changes from 3 to 6, and we set the dependency between subtasks based on the length of the waiting time. In addition, the number of skills required for each subtask is randomly selected within the range of $[sk^-, sk^+]$, changing from $[1, 2]$ to $[1, 5]$. Moreover, the start time a and waiting time wt of each task are randomly distributed within the range of $[0, 20]$ and $[20, 30]$, respectively. For workers, the number of workers n varies from 500 to 2000, and the number of skills sk mastered by each worker is randomly selected within the range of $[1, 2]$. Additionally, we generate the moving speed v of each worker to be randomly distributed in the range of $[0.001, 0.01]$ and set the cost of moving unit distance c as 10. Moreover, the maximum moving distance d of each worker varies from 0.05 to 0.4. For both workers and tasks, the size of the skill universe $|s|$ is set to 70.

7.2. Approaches

In this subsection, we introduce the approaches applied to address the MSCTA problem. We compare the greedy algorithm and the game algorithm with a baseline algorithm, called the MSSCG algorithm, which refers to the MSSC greedy algorithm in [12].

Specifically, in each round of the greedy algorithm, the platform selects the best worker for each subtask according to the assignment strategies mentioned in Section 5. However, the achieved assignment result is a local optimal solution, all matching pairs are determined by the platform, and workers cannot choose independently. By contrast, in each round of the game algorithm, workers can randomly select the best subtask according to their preferences until the algorithm reaches a Nash equilibrium. Since we apply the greedy algorithm to initialize the result of the game algorithm, we abbreviate the game algorithm as G G in our experiments. In addition, similar to the greedy algorithm, the MSSC G algorithm selects the worker with the highest profit increase in each round, but ignores the dependency constraints between tasks, and does not consider whether the task can meet the completion conditions.

In our experiments, we only change one parameter at a time, and the remaining parameters are set to the default values. All of our experiments were run on an Intel Core i5-8400 CPU @2.80 GHz with 8 GB RAM.

Table 6
Parameter Information of Synthetic Data.

Parameters	Values
the number of stages, l	3, 4, **5**, 6
the number of skills for subtasks, $\|sk\|$	[1, 2], **[1, 3]**, [1, 4], [1, 5]
the number of subtasks, ml	300, 600, **900**, 1200
the number of workers, n	500, 1000, **1500**, 2000
the maximum moving distance, d	0.05, **0.1**, 0.2, 0.3, 0.4

7.3. Experiments

7.3.1. Experiments on synthetic data set

For synthetic data, we analyze the effect of the following parameters on experimental results: the number of task stages l, the number of skills $|sk|$ of each subtask, the number of subtasks ml, the number of workers n, and the maximum moving distance d of workers.

Effect of the number of stages l. In Fig. 2(a), when l increases, the profits obtained by our algorithms decrease. With increasing l, the number of executable subtasks included in ET decreases, resulting in fewer subtasks available for matching in each round, and making many subtasks unable to be effectively matched due to the dependency constraint. In addition, we obtain that the profit obtained by G G is more than the profit obtained by the greedy algorithm. Moreover, the profit obtained by the greedy algorithm is smaller than that obtained by the MSSC G algorithm at first, and then is greater than that obtained by the MSSC G algorithm when l is greater than 4.

In Fig. 2(b), with the increase of l, the running time of G G and the greedy algorithm decreases. Due to the fewer subtasks included in ET, the time complexity of each round in our algorithms is lower, and the number of subtasks for valid matching will be reduced due to the constraint of dependency. Since the MSSC G algorithm does not consider the dependency constraint, its running time remains unchanged.

Effect of the number of skills of each subtask. Fig. 3 presents the effect of the number of skills $|sk|$ on the results. It is observed from Fig. 3(a) that when $\left[|sk|^-, |sk|^+\right]$ changes from $[1,2]$ to $[1,5]$, the profits obtained by our algorithms decrease. The reason is that the increase in $\left[|sk|^-, |sk|^+\right]$ makes it increasingly more difficult to meet the skill constraints of the subtasks. In addition, it is observed from the figure that the profit obtained by G G is greater than the profits obtained by the greedy algorithm and the MSSC G algorithm.

In Fig. 3(b), with increasing $\left[|sk|^-, |sk|^+\right]$, the running time increases for all of the algorithms. This is because with increasing $\left[|sk|^-, |sk|^+\right]$, each subtask needs to be matched with more workers to complete, making the MSCTA problem more complex and requiring more running time.

Effect of the number of subtasks ml. It is observed from Fig. 4(a) that the profits obtained by our methods increase when ml increases from 300 to 1200. The reason is that the increase in ml allows more subtasks to be assigned to the workers, resulting in higher profits. In addition, the profit obtained by G G is the most, followed by the greedy algorithm, and the MSSC G algorithm gets the least profit.

As shown in Fig. 4(b), the running time of each algorithm increases with increasing ml. This is because the more subtasks cause the MSCTA problem to be more complicated, and more time is required to complete the matching of tasks and workers. Moreover, the running time of the MSSC G algorithm is shorter than that of G G but is longer than that of the greedy algorithm.

Effect of the number of workers n. Fig. 5 shows the impact of n changing from 500 to 2000. As observed from Fig. 5(a), the profit obtained by our approaches increases with increasing n. The reason is that when n increases, each subtask has more options to match better workers. However, because the number of subtasks is given, we can deduce that when n increases to complete all subtasks, the profit will remain stable. Compared with G G, the greedy algorithm obtains lower profit. In addition, the profit obtained by the greedy algorithm is almost equal to that obtained by the MSSC G algorithm at first, and then is greater than that obtained by the MSSC G algorithm when n is greater than 1500.

An examination of Fig. 5(b) shows that the running time of all algorithms increases with increasing n, and the reason for this trend is similar to that of the trend observed in Fig. 4(b). The greedy algorithm has the least running time, followed by the MSSC G algorithm, and G G has the longest running time.

Effect of the maximum moving distance of workers d. As shown in Fig. 6(a), when d increases from 0.05 to 0.2, the profits obtained by our algorithms increase, and then when d further increases from 0.2 to 0.4, the profits tend to be stable. The reason is that, when d increases from 0.05 to 0.2, each subtask has more options to match better workers, so the profit

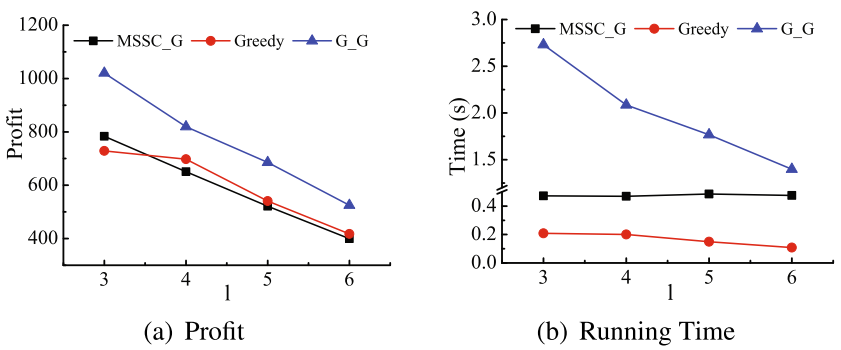

Fig. 2. Effect of the number of stages.

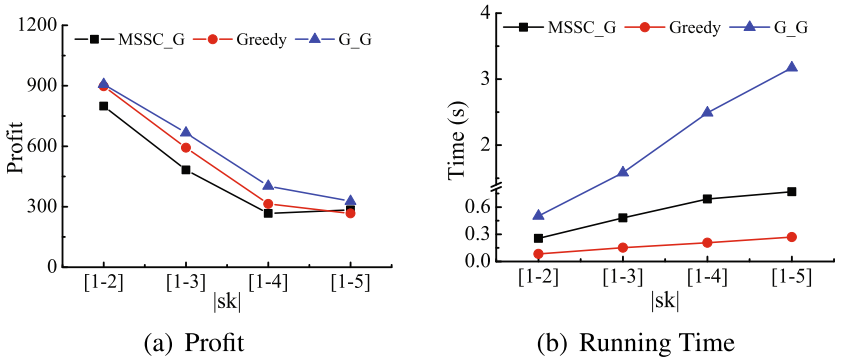

Fig. 3. Effect of the number of skills.

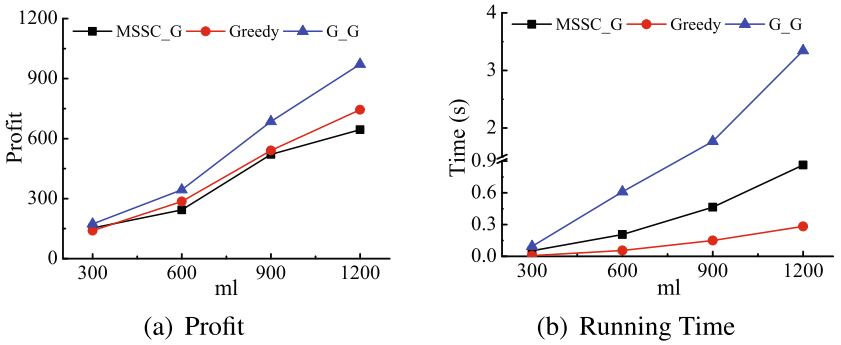

Fig. 4. Effect of the number of subtasks.

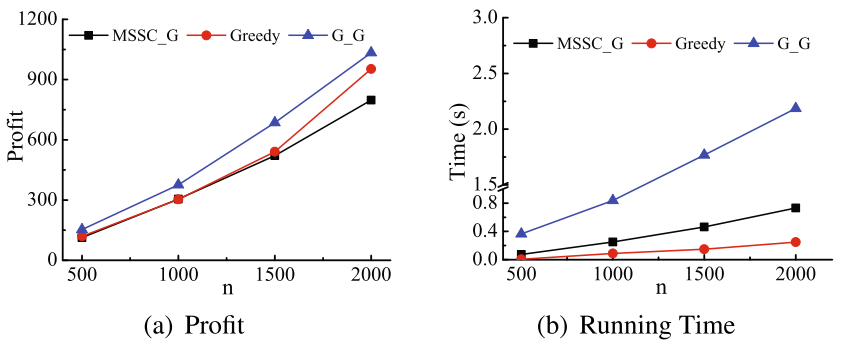

Fig. 5. Effect of the number of workers.

increases. However, since the number of subtasks has been given, the number of tasks that each worker can choose will not continue to increase with increasing d, so the profit remains stable when d further increases from 0.2 to 0.4. In addition, G G obtains the most profit, followed by the greedy algorithm, and the MSSC G algorithm obtained the least profit.

It is observed from Fig. 6(b) that the running time of our algorithms first increases with increasing d and then tends to be stable. The reason for this trend is similar to that observed in Fig. 6(a). When d increases from 0.05 to 0.2, each subtask has more options to match better workers, so the MSCTA problem becomes more complicated, and the running time of our algorithms increases. However, when d further increases from 0.2 to 0.4, the number of subtasks that each worker can choose will not continue to increase with increasing d, so the running time of our algorithms remains stable.

7.3.2. Experiments on real data sets

For the real data set from [41], we analyze the effect of the budget range $[b^-, b^+]$ and the range of start time $[a^-, a^+]$ on experimental results.

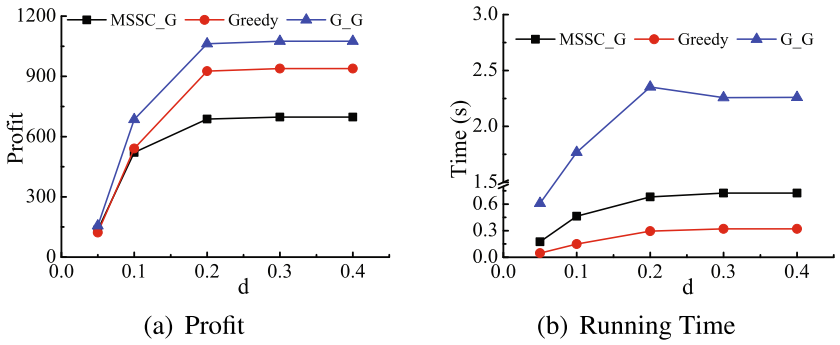

Fig. 6. Effect of the maximum moving distance.

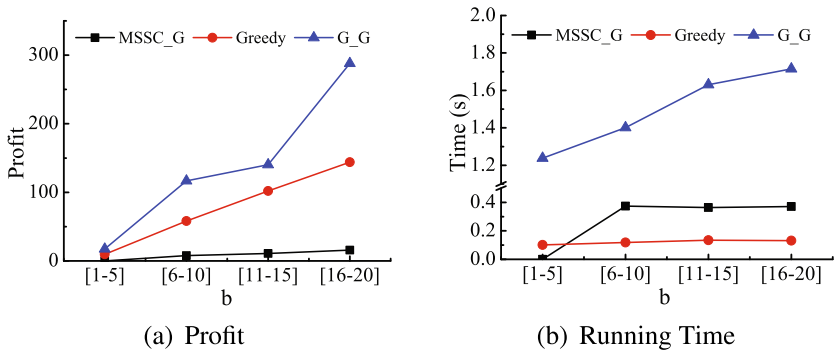

Fig. 7. Effect of the budget range.

Effect of the budget range $[b^-, b^+]$. Fig. 7 illustrates the influence of the budget range $[b^-, b^+]$ from $[1, 5]$ to $[16, 20]$. It is observed from Fig. 7(a) that the profits obtained by all algorithms increase with the increase of the budget. This is because the increase in $[b^-, b^+]$ can attract more workers to complete tasks and generate more profit. Moreover, we also find that the profit obtained by the greedy algorithm is greater than that obtained by the MSSC G algorithm, but is less than that obtained by G G.

An examination of the results presented in Fig. 7(b) shows that the running time of G G increases obviously with the increase of $[b^-, b^+]$, while the running time of the greedy algorithm increases slightly. The reason is that, with an increase in $[b^-, b^+]$, each subtask has more options to match better workers, making the MSCTA problem more complex and increasing the running time of our algorithms. Moreover, the greedy algorithm has the least running time, followed by the MSSC G algorithm, and the running time of G G is the most.

Effect of the range of start time $[a^-, a^+]$. Fig. 8 depicts the experimental results of $[a^-, a^+]$ from $[0, 5]$ to $[0, 20]$. In Fig. 8(a), with the increase of $[a^-, a^+]$, the profit received by our algorithms decreases. Because the increase in $[a^-, a^+]$ disperses the

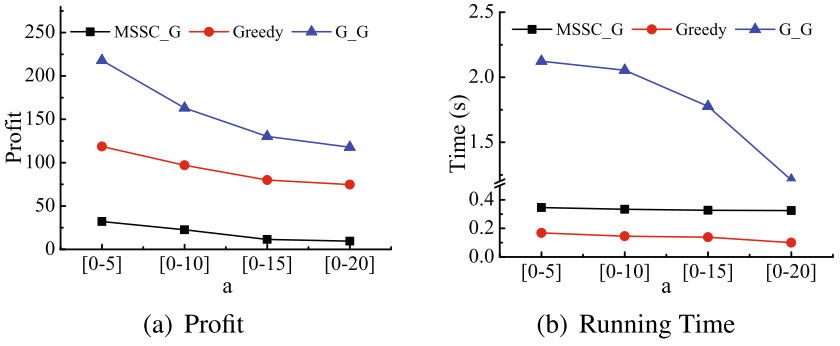

Fig. 8. Effect of the range of start time.

distribution of subtasks and workers over time, it is difficult to match enough workers to complete the subtasks. Moreover, it is observed that G G obtains the highest profit, followed by the greedy algorithm, and the MSSC G algorithm gets the least profit.

An examination of Fig. 8(b) shows that as $[a^-, a^+]$ increases, the running time of G G decreases. The reason is similar to that for the results presented in Fig. 8(a); namely, the increase in $[a^-, a^+]$ disperses the distribution of the subtasks and workers over time, which implies that the strategic space of each worker becomes smaller and that the complexity of the MSCTA problem is reduced, thereby reducing the running time of our algorithms. In addition, we observed that the running time of the MSSC G algorithm is more than that of the greedy algorithm.

For the real data set from [3], we analyze the effect of the number of subtasks *ml* and the number of the workers *n* on experimental results.

Effect of the number of subtasks ml. It is observed from Fig. 9(a) that the profit obtained by our methods increases when *ml* increases from 5000 to 10000. The profit obtained by G G is the most, followed by the greedy algorithm, and the MSSC G algorithm gets the least profit. The reason is similar to that for the results presented in Fig. 4(a).

As shown in Fig. 9(b), the running time of each algorithm increases with increasing *ml*. The running time of the MSSC G algorithm is shorter than that of G G but is longer than that of the greedy algorithm.

Effect of the number of workers n. Fig. 10 shows the impact of *n* changing from 3000 to 9000. As observed from Fig. 10(a), the profit obtained by our approaches increases with increasing *n*, and then gradually stabilizes. The reason is the same as described in Fig. 5(a). Compared with G G, the greedy algorithm obtains a lower profit, but its profit is greater than that of the MSSC G algorithm.

An examination of Fig. 10(b) shows that the running time of all algorithms increases with increasing *n*. The greedy algorithm has the least running time, followed by the MSSC G algorithm, and G G has the longest running time.

According to the results of the above experiments, we summarize our findings as follows: 1) G G obtains greater profit than the greedy algorithm and the MSSC G algorithm but requires the longest running time; and 2) the profit obtained by the greedy algorithm is less than that obtained by G G, but its running time is the shortest among all algorithms.

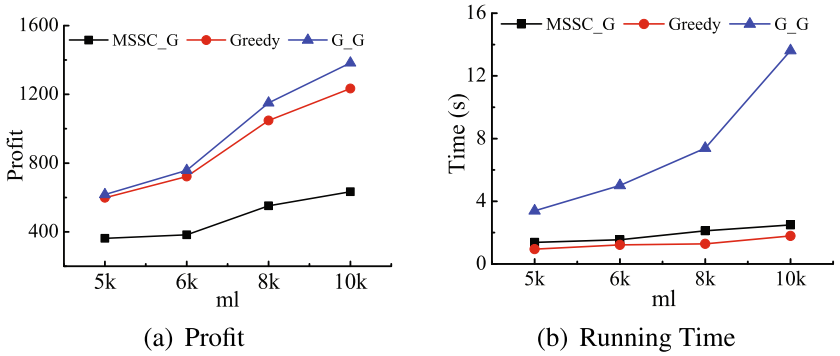

Fig. 9. Effect of the number of subtasks.

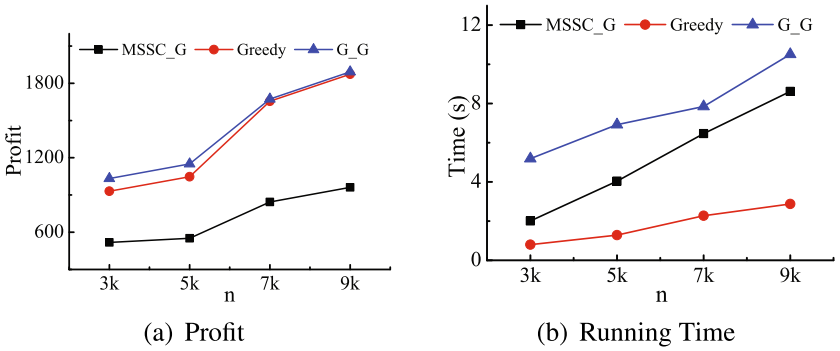

Fig. 10. Effect of the number of workers.

8. Conclusion

In this paper, we propose a new problem in spatial crowdsourcing, that is, the multi-stage complex task assignment (MSCTA) problem, where each task is composed of multiple complex subtasks (or stages) with dependencies. The purpose of the MSCTA problem is to allocate workers to multi-stage complex tasks to gain the maximal total profit. In addition to the constraint of dependency, this problem is also constrained by the maximal working area, skills, budget, and deadline. We first demonstrate that the MSCTA problem is NP-hard and then propose a greedy algorithm and a game algorithm to solve this problem. Finally, we have conducted many experiments to demonstrate the efficiency of our algorithms. In future work, we will further study the influence of workers' preferences on the MSCTA problem, and consider how to combine with the proposed algorithms when multiple preferences exist at the same time. In addition, we also plan to schedule workers in advance by predicting where and when tasks will appear, so that tasks can be completed more efficiently.

CRediT authorship contribution statement

Zhao Liu: Conceptualization, Methodology, Writing - original draft. **Kenli Li:** Supervision, Funding acquisition, Writing - review & editing. **Xu Zhou:** Data curation, Validation. **Ningbo Zhu:** Visualization. **Yunjun Gao:** Investigation. **Keqin Li:** Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant Nos.61772182, 62172146, 61802032, 62172157), Zhejiang Lab (Grant No. 2021KD0AB02), and the Project of HuNan Science and Technology Innovation Plan (2020RC2032).

References

[1] Lei Chen, Cyrus Shahabi, Spatial crowdsourcing: Challenges and opportunities, IEEE Data Eng. Bull. 39 (4) (2016) 14–25.
[2] Meituan (2020). https://www.meituan.com/. Accessed September 4, 2020..
[3] Didichuxing (2020). https://www.didiglobal.com/. Accessed September 4, 2020..
[4] Taskrabbit (2020). http://www.taskrabbit.com/. Accessed September 4, 2020..
[5] Yan Liu, Bin Guo, He Du, Zhiwen Yu, Daqing Zhang, and Chao Chen. Poster: Foodnet: Optimized on demand take-out food delivery using spatial crowdsourcing. In Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking, pages 564–566, 2017..
[6] Libin Zheng, Lei Chen, Multi-campaign oriented spatial crowdsourcing, in: 2018 IEEE 34th International Conference on Data Engineering (ICDE), 2018, pp. 1248–1251.
[7] Y. Li, J. Gao, P.P.C. Lee, L. Su, C. He, C. He, F. Yang, W. Fan, A weighted crowdsourcing approach for network quality measurement in cellular data networks, IEEE Transactions on Mobile Computing 16 (2) (2017) 300–313.
[8] Habibur Rahman, Saravanan Thirumuruganathan, Senjuti Basu Roy, Sihem Amer-Yahia, and Gautam Das. Worker skill estimation in team-based tasks. Proceedings of the VLDB Endowment, 8(11), 1142–1153, 2015..
[9] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, Stefano Leonardi, Online team formation in social networks, in: Proceedings of the 21st international conference on World Wide Web, 2012, pp. 839–848.
[10] Theodoros Lappas, Kun Liu, Evimaria Terzi, Finding a team of experts in social networks, in: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, 2009, pp. 467–476.
[11] Tianshu Song, Xu. Ke, Jiangneng Li, Yiming Li, Yongxin Tong, Multi-skill aware task assignment in real-time spatial crowdsourcing, GeoInformatica 24 (1) (2020) 153–173.
[12] Peng Cheng, Xiang Lian, Lei Chen, Jinsong Han, Jizhong Zhao, Task assignment on multi-skill oriented spatial crowdsourcing, IEEE Transactions on Knowledge and Data Engineering 28 (8) (2016) 2201–2215.
[13] Wangze Ni, Peng Cheng, Lei Chen, Xuemin Lin, Task allocation in dependency-aware spatial crowdsourcing, in: 2020 IEEE 36th International Conference on Data Engineering (ICDE), 2020, pp. 985–996.
[14] Decui Liang, Wen Cao, Xu. Zeshui, Mingwei Wang, A novel approach of two-stage three-way co-opetition decision for crowdsourcing task allocation scheme, Information Sciences 559 (2021) 191–211.
[15] Xiaoyu Zhang, Xiaofeng Chen, Hongyang Yan, Yang Xiang, Privacy-preserving and verifiable online crowdsourcing with worker updates, Information Sciences 548 (2021) 212–232.
[16] Xu. Wenqiang, Liangxiao Jiang, Chaoqun Li, Improving data and model quality in crowdsourcing using cross-entropy-based noise correction, Information Sciences 546 (2021) 803–814.
[17] Yongxin Tong, Zimu Zhou, Yuxiang Zeng, Lei Chen, Cyrus Shahabi, Spatial crowdsourcing: a survey, The VLDB Journal 29 (1) (2020) 217–250.
[18] Xiaohui Bei and Shengyu Zhang. Algorithms for trip-vehicle assignment in ride-sharing. In Thirty-Second AAAI Conference on Artificial Intelligence, pages 3–9, 2018..
[19] Shibo He, Dong-Hoon Shin, Junshan Zhang, Jiming Chen, Toward optimal allocation of location dependent tasks in crowdsensing, in: IEEE INFOCOM 2014-IEEE Conference on Computer Communications, 2014, pp. 745–753.
[20] Zhao Chen, Peng Cheng, Yuxiang Zeng, Lei Chen, Minimizing maximum delay of task assignment in spatial crowdsourcing, in: 2019 IEEE 35th International Conference on Data Engineering (ICDE), 2019, pp. 1454–1465.
[21] Mohammad Asghari, Cyrus Shahabi, On on-line task assignment in spatial crowdsourcing, in: 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 395–404.
[22] Leyla Kazemi, Cyrus Shahabi, Geocrowd: enabling query answering with spatial crowdsourcing, in: Proceedings of the 20th international conference on advances in geographic information systems, 2012, pp. 189–198.

[23] Peng Cheng, Xiang Lian, Zhao Chen, Rui Fu, Lei Chen, Jinsong Han, and Jizhong Zhao. Reliable diversity-based spatial crowdsourcing by moving workers. Proc. VLDB Endow., pages 1022–1033, 2015..
[24] Dawei Gao, Yongxin Tong, Jieying She, Tianshu Song, Lei Chen, Xu. Ke, Top-k team recommendation and its variants in spatial crowdsourcing, Data Science and Engineering 2 (2) (2017) 136–150.
[25] Dingxiong Deng, Cyrus Shahabi, Ugur Demiryurek, Maximizing the number of worker's self-selected tasks in spatial crowdsourcing, in: Proceedings of the 21st ACM sigspatial international conference on advances in geographic information systems, 2013, pp. 324–333.
[26] Y. Liu, B. Guo, C. Chen, H. Du, Z. Yu, D. Zhang, H. Ma, Foodnet: Toward an optimized food delivery network based on spatial crowdsourcing, IEEE Transactions on Mobile Computing 18 (6) (2019) 1288–1301.
[27] Yan Zhao, Yu. Yang Li, Han Su Wang, Kai Zheng, Destination-aware task assignment in spatial crowdsourcing, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 297–306.
[28] Yan Zhao, Kai Zheng, Yang Li, Han Su, Jiajun Liu, and Xiaofang Zhou. Destination-aware task assignment in spatial crowdsourcing: A worker decomposition approach. IEEE Transactions on Knowledge and Data Engineering, pages 2336–2350, 2020..
[29] Yi Xu, Yongxin Tong, Yexuan Shi, Qian Tao, Ke Xu, and Wei Li. An efficient insertion operator in dynamic ridesharing services. In 2019 IEEE 35th International Conference on Data Engineering (ICDE), pages 1022–1033, 2019..
[30] Peng Cheng, Lei Chen, Jieping Ye, Cooperation-aware task assignment in spatial crowdsourcing, in: 2019 IEEE 35th International Conference on Data Engineering (ICDE), 2019, pp. 1442–1453.
[31] L. Wang, Z. Yu, Q. Han, B. Guo, H. Xiong, Multi-objective optimization based allocation of heterogeneous spatial crowdsourcing tasks, IEEE Transactions on Mobile Computing 17 (7) (2018) 1637–1650.
[32] M. Xiao, K. Ma, A. Liu, H. Zhao, Z. Li, K. Zheng, X. Zhou, Sra: Secure reverse auction for task assignment in spatial crowdsourcing, IEEE Transactions on Knowledge and Data Engineering 32 (4) (2020) 782–796.
[33] L. Zheng, L. Chen, Multi-campaign oriented spatial crowdsourcing, IEEE Transactions on Knowledge and Data Engineering 32 (4) (2020) 700–713.
[34] Vijay V. Vazirani, Approximation algorithms, Springer, 2001.
[35] George L Nemhauser, Laurence A Wolsey, Marshall L Fisher, An analysis of approximations for maximizing submodular set functions, Mathematical programming 14 (1) (1978) 265–294.
[36] Chubo Liu, Kenli Li, Zhuo Tang, Keqin Li, Bargaining game-based scheduling for performance guarantees in cloud computing, ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) 3 (1) (2018) 1–25.
[37] Zheng Xiao, Zhao Tong, Kenli Li, Keqin Li, Learning non-cooperative game for load balancing under self-interested distributed environment, Applied Soft Computing 52 (2017) 376–386.
[38] J. Hu, K. Li, C. Liu, K. Li, A game-based price bidding algorithm for multi-attribute cloud resource provision, IEEE Transactions on Services Computing (2018) 1.
[39] Zhao Liu, Xu. Kenli Li, Ningbo Zhu Zhou, Keqin Li, Incentive mechanisms for crowdsensing: Motivating users to preprocess data for the crowdsourcer, ACM Transactions on Sensor Networks (TOSN) 16 (4) (2020) 1–24.
[40] Dov Monderer, Lloyd S. Shapley, Potential games, Games and Economic Behavior 14 (1) (1996) 124–143.
[41] Meetup (2020). https://www.meetup.com/. Accessed September 4, 2020..

Zhao Liu is currently working for a Ph.D. degree in computer science and technology with the College of Information Science and Engineering, Hunan University, Changsha, China. His research interests include spatio-temporal data management and AI accelerators.

Kenli Li received the PhD degree in computer science from Huazhong University of Science and Technology, China, in 2003. His major research interests include parallel computing, highperformance computing, and grid and cloud computing. He has published more than 200 research papers in international conferences and journals such as the IEEE Trans. On Computers, IEEE Trans. on Parallel and Distrib. Syst., and ICPP. He is a senior member of the IEEE and serves on the editorial board of the IEEE-TC.

Xu Zhou received the Ph.D. degree in computer science and technology from Hunan University, China, in 2016. She is currently an Associate Professor with the College of Computer Science and Electronic Engineering, Hunan University. She has published over 30 papers in international journals and conferences, such as the IEEE Trans. on Knowl. and Data Eng., the IEEE Trans. Parallel Distrib. Syst., and the IEEE Trans. on Computers. Her current research interests include parallel and distributed processing, and database systems.

Ningbo Zhu received a Ph.D. degree in engineering from the computer use and application, Nanjing University of Science and Technology, Nanjing, China, in 2005. He is a Professor at the College of Information Science and Engineering, Hunan University, Changsha, China. His research interests include pattern recognition and intelligent system, digital image processing and network and information security.

Yunjun Gao received the PhD degree in computer science from Zhejiang University, China, in 2008. He is currently a professor with the College of Computer Science, Zhejiang University, China. His research interests include spatial and spatio-temporal databases, metric and incomplete/uncertain data management, and spatio-textual data processing. He is a member of the ACM and the IEEE, and a senior member of the CCF.

Keqin Li is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a National Distinguished Professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyberphysical systems, heterogeneous computing systems, big data computing, high performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 810 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds nearly 60 patents announced or authorized by the Chinese National Intellectual Property Administration. He has chaired many international conferences. He is currently an associate editor of the ACM Computing Surveys and the CCF Transactions on High Performance Computing. He has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.