The VLDB Journal (2024) 33:163–184
https://doi.org/10.1007/s00778-023-00802-3

REGULAR PAPER

Coalition-based task assignment with priority-aware fairness in spatial crowdsourcing

Yan Zhao[1] · Kai Zheng[2] · Ziwei Wang[3] · Liwei Deng[3] · Bin Yang[1] · Torben Bach Pedersen[1] · Christian S. Jensen[1] · Xiaofang Zhou[4]

Received: 13 July 2022 / Revised: 15 May 2023 / Accepted: 19 June 2023 / Published online: 7 July 2023

Abstract
With the widespread use of networked and geo-positioned mobile devices, e.g., smartphones, Spatial Crowdsourcing (SC), which refers to the assignment of location-based tasks to moving workers, is drawing increasing attention. One of the critical issues in SC is task assignment that allocates tasks to appropriate workers. We propose and study a novel SC problem, namely Coalition-based Task Assignment (CTA), where the spatial tasks (e.g., home improvement and furniture installation) may require more than one worker (forming a coalition) to cooperate to maximize the overall rewards of workers. We design a greedy and an equilibrium-based CTA approach. The greedy approach forms a set of worker coalitions greedily for performing tasks and uses an acceptance probability to identify high-value task assignments. In the equilibrium-based approach, workers form coalitions in sequence and update their strategies (i.e., selecting a best-response task), to maximize their own utility (i.e., the reward of the coalition they belong to) until a Nash equilibrium is reached. Since the equilibrium obtained is not unique and optimal in terms of total rewards, we further propose a simulated annealing scheme to find a better Nash equilibrium. To achieve fair task assignments, we optimize the framework to distribute rewards fairly among workers in a coalition based on their marginal contributions and give workers who arrive first at the SC platform highest priority. Extensive experiments demonstrate the efficiency and effectiveness of the proposed methods on real and synthetic data.

Keywords Coalition · Task assignment · Spatial crowdsourcing · Priority-aware fairness

1 Introduction

Spatial Crowdsourcing (SC) enables people to move as multi-modal sensors that collect and share instantaneously various types of high-fidelity spatio-temporal data. Specifically, task requesters can issue spatial tasks, which require the presence of workers at specified physical locations, to an SC server that then assigns workers carrying smart devices to travel to the specified locations and perform the tasks. This is referred to as *task assignment*.

✉ Kai Zheng
zhengkai@uestc.edu.cn

Yan Zhao
yanz@cs.aau.dk

Ziwei Wang
ziwei@std.uestc.edu.cn

Liwei Deng
deng_liwei@std.uestc.edu.cn

Bin Yang
byang@cs.aau.dk

Torben Bach Pedersen
tbp@cs.aau.dk

Christian S. Jensen
csj@cs.aau.dk

Xiaofang Zhou
zxf@cse.ust.hk

[1] Department of Computer Science, Aalborg University, Aalborg, Denmark

[2] Yangtze Delta Region Institute (Quzhou), School of Computer Science and Engineering, Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, Chengdu, China

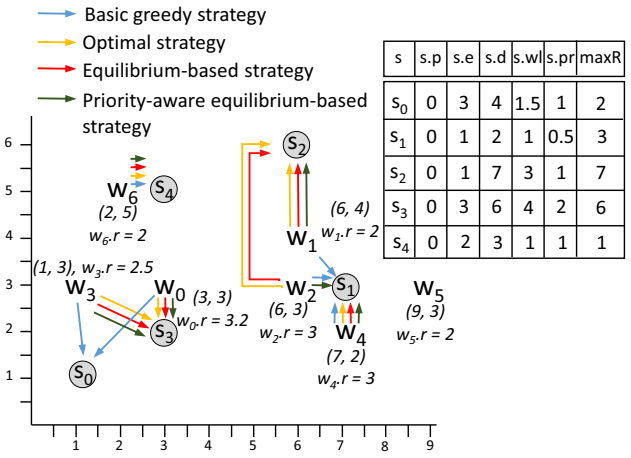
[3] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China

[4] The Hong Kong University of Science and Technology, Hong Kong, China

A number of existing studies place their focus on "single" task assignment, where a task is assigned to a single worker [7, 15, 27, 29, 38, 42, 44]. However, SC applications exist in which a single worker cannot efficiently perform a task, e.g., home improvement, furniture installation, or performing entertainment at an event [2, 3, 34]. Instead, workers have to form a coalition to jointly complete these complex tasks that exceed the capabilities of an individual worker. Moreover, a worker may prefer to collaborate with other workers for reputation or economic purposes.

We investigate SC task assignment in this problem setting, namely Coalition-based Task Assignment (CTA). Given a set of workers and a set of tasks, the objective is to assign a stable worker coalition to each task to achieve the highest total reward. The proposed coalition-based task assignment can be applied in real-world scenarios such as home improvement, furniture installation, monitoring of traffic conditions in an area, organizing an event, or performing entertainment at an event [2, 3, 34], where workers have to collaborate to jointly complete complex tasks that exceed the capabilities of an individual worker. Some recent studies have explored task assignment approaches that allow each task to be assigned to multiple workers [16, 18, 19, 37, 43]. But workers can conduct tasks independently and without cooperating, which is different from our problem. The most related study [2] concerns collaboration-aware task assignment, where workers are required to cooperate and accomplish tasks jointly for achieving high total cooperation quality scores. However, this approach assumes that workers always conduct tasks voluntarily, which may be unrealistic unless workers are rewarded for doing so. Further, this study aims to find a single Nash equilibrium point and does not consider whether more optimal equilibrium points exist. We target instead a setting in which workers are rewarded if they cooperate to complete tasks, and we aim to achieve a Nash equilibrium with high total reward.

We illustrate the CTA problem through the example in Fig. 1, which involves seven workers ($\{w_0, ..., w_6\}$) and five tasks ($\{s_0, ..., s_4\}$). Each worker has a current location and a reachable distance (marked as $w.r$). A task is published at a time instance and expires at a time instance, and it is labeled with a workload ($s.wl$) and reward information (i.e., penalty rate $s.pr$ and maximum reward $s.maxR$). The reward for completing a task is obtained using a reward pricing model, to be covered in Sect. 2. The problem is to assign tasks to workers so as to maximize the total reward. For the sake of simplicity, we assume that all workers share the same speed (1) in this example and that their travel times between locations can be estimated using Euclidean distances. We also assume that the online time of workers are 0. In SC, it is intuitive to assign nearby workers to tasks greedily (in order to obtain the maximal actual reward for each task) without violating the spatio-temporal constraints (i.e., assigned



s	s.p	s.e	s.d	s.wl	s.pr	maxR
s_0	0	3	4	1.5	1	2
s_1	0	1	2	1	0.5	3
s_2	0	1	7	3	1	7
s_3	0	3	6	4	2	6
s_4	0	2	3	1	1	1

Fig. 1 Running example

tasks should be located in the reachable ranges of the corresponding workers, and workers must arrive at the locations of assigned tasks before the deadline of tasks), referred to as the basic greedy algorithm. With this algorithm, we obtain the assignment, $\{(s_0, \{w_0, w_3\}), (s_1, \{w_1, w_2, w_4\}), (s_4, \{w_6\})\}$ (shown with blue arrows in Fig. 1), with the overall reward of 5.76. This assignment leaves s_2 and s_3 (that have high rewards) unassigned, which decreases the overall reward. Applying the optimal strategy, we can achieve the assignment, $\{(s_1, \{w_4\}), (s_2, \{w_1, w_2\}), (s_3, \{w_0, w_3\}), (s_4, \{w_6\})\}$ (shown with yellow arrows in the figure), the total reward of which is 12.26. However, existing optimal task assignment algorithms have high computational costs and do not scale to support practical applications [30, 42].

In the setting we study, workers must form coalitions with sufficient cumulative time or capabilities across the coalition members to accomplish the tasks. To enable this, we propose two novel coalition-based task assignment algorithms, i.e., greedy and equilibrium-based algorithms that aim to achieve high total rewards. Specifically, the greedy algorithm is a non-reducing reward allocation strategy that incentivizes workers to enlarge a coalition to achieve a higher total reward. By considering the time utilization ratio (measured by workers' workload and travel time) of workers and the rate of return (measured by tasks' actual rewards and maximal reward) of tasks, the greedy algorithm adopts an acceptance probability to find high-value task assignments (i.e., task assignments ensuring high time utilization ratios for workers and high rates of return for tasks). However, the greedy approach cannot guarantee the stability of formed coalitions. Therefore, we propose a task assignment method with a game-theoretic basis. We convert the CTA problem into a multi-player game, prove that it is an exact potential game that has multiple pure Nash equilibriums (cf. Definition 7 and Lemmas 3–4) and develop an equilibrium-based algorithm that applies the best-response method with sequential and asynchronous updates

of workers' strategies to reach a pure Nash equilibrium. The benefit is that if workers are all in a Nash equilibrium, they are closely inter-connected, and the formed coalitions are stable, i.e., no worker can improve their utility by a unilateral change to a different coalition if other workers stay in their coalitions. The task assignment generated by the best-response method is locally optimal, and there may exist multiple Nash equilibriums since the CTA game is an exact potential game. We therefore propose a simulated annealing optimization strategy that can coordinate workers to obtain a better Nash equilibrium. In Fig. 1, the overall reward generated by the equilibrium-based algorithm (red arrows) is 12.26, which is the maximal reward. This is so because there is only one equilibrium in this small-scale example.

Although the above algorithms achieve effect task assignment, they ignore the aspect of fairness in task assignments. Fairness, which is crucial in SC, is measured in terms of the correlation between the contribution and the reward of a worker. In a fair assignment, a worker should receive a reward proportional to the worker's (non-negative) contribution to the coalition. The rationale is that if a worker is not compensated for their contribution then this worker is being "underpaid" or not recognized. Also, an "unfair" compensation may incentivize a worker to leave the coalition, in which case the reward of the entire coalition may decrease. Moreover, workers may recognize unfair treatment and may avoid such cases. Further, if a worker is treated unfairly, the worker will lose trust in the SC platform and may refuse further participation. To address this issue, we optimize the original framework by take fairness into account. Specifically, we first introduce the marginal contribution-based value to explore the reward distribution mechanisms for workers with varying marginal contributions (that are used to measure the contributions of workers in a coalition). The marginal contribution-based value distributes rewards fairly among workers due to its relevant interpretation of marginal contributions. When the workloads and assignment orders of workers in a coalition differ, the marginal contributions of workers change, and so do the rewards of workers. In addition, we propose the concept of priority-aware fairness giving workers who arrive first at the SC platform highest priority, where a priority-aware utility (PAU) is used to improve the greedy and equilibrium-based task assignment approaches to achieve their optimization goal of maximizing the total reward with priority-aware fairness as a constraint. In Fig. 1, the priority-aware equilibrium-based strategy can generate the assignment, $\{(s_1, \{w_2, w_4\}), (s_2, \{w_1\}), (s_3, \{w_0, w_3\}), (s_4, \{w_6\})\}$. The total reward is 11.51, which is 93.89% of that achieved by the optimal and the equilibrium-based strategies. However, the payoff difference (see Eq. 22) among workers generated by the priority-aware equilibrium-based strategy is only 0.11 while the payoff difference of the optimal and the equilibrium-based strategies is 0.14. Apparently, it is more fair if the payoff difference among workers is smaller, and the priority-aware equilibrium-based strategy can achieve more fair task assignment.

The contributions can be summarized as follows:

1) We formulate a novel task assignment in SC, namely Coalition-based Task Assignment (CTA), where workers need to interact with others by forming worker coalitions to conduct the corresponding tasks.
2) A greedy approach is developed to efficiently assign tasks, in which an acceptance probability is introduced to find the high-value task assignments.
3) We develop an equilibrium-based solution, wherein the Nash equilibrium is found based on the best-response approach. Inspired by the success of simulated annealing in finding a better Nash equilibrium in the best response dynamic [21], we use a simulated annealing strategy to further improve the assignment when multiple Nash equilibriums exist.
4) We introduce the marginal contribution-based value to compute each worker's reward in a coalition fairly and formalize the concept of priority-aware fairness. We redesign the greedy and equilibrium-based task assignment approaches to achieve the optimization goal of maximizing the total reward with priority-aware fairness as a constraint.
5) Extensive experiments are conducted to study the impact of the key parameters and the effectiveness of the proposed algorithms. The remainder of this paper is organized as follows. Preliminary concepts and the Reward Pricing Model are introduced in Sect. 2. We then present the greedy task allocation algorithm and the equilibrium-based algorithm in Sect. 3 and Sect. 4, respectively, followed by the fairness extension in Sect. 5. We report the experimental results in Sect. 6. Section 7 surveys the related work, and Sect. 8 offers conclusions.

2 Problem statement

We proceed to first introduce a set of preliminary concepts in the context of reward-based multiple task assignment, where a task can be assigned to multiple workers who will each receive a reward for the task, in spatial crowdsourcing. Then, we present the reward pricing model for tasks and formulate our problem. Table 1 lists notation used throughout the paper.

Definition 1 (*Spatial Task*) A spatial task, denoted as $s = (l, p, e, d, wl, maxR, pr)$, has a location $s.l$, a publication time $s.p$, an expected completion time $s.e$, and a deadline $s.d$. Each task is also labeled with a required workload $s.wl$ incurred in order to finish the task (we simply use the time

Table 1 Summary of notation

Symbol	Definition
s	Spatial task
$s.l$	Location of spatial task s
$s.p$	Publication time of spatial task s
$s.e$	Expected completion time of spatial task s
$s.d$	Deadline of spatial task s
$s.wl$	Workload of spatial task s
$s.maxR$	Maximum reward of spatial task s
$s.pr$	Penalty rate of spatial task s
w	Worker
$w.l$	Location of worker w
$w.on$	Online time of worker w
$w.sp$	Speed of worker w
$w.r$	Reachable distance of worker w
$AW(s)$	Available worker set of task s
t_{now}	The current time
$t(a, b)$	Travel time from location a to b
$d(a, b)$	Travel distance from location a to b
$RS(w)$	Reachable task set of worker w
$WC(s)$	Worker coalition for task s
$R_{WC(s)}$	Reward of $WC(s)$ by finishing s
$R_{MWC(s)}$	Reward of $MWC(s)$ by finishing s
$s.t_e$	The completion time of s
$s.t_s$	Assignment time of task s
$T_{WC(s)}$	The duration of task s given $WC(s)$
$T_{MWC(s)}$	The duration of task s given $MWC(s)$
$w.wl(WC(s))$	The workload of w performing s in $WC(s)$
$MWC(s)$	Minimal worker coalition for task s
A	A spatial task assignment
$A.R$	Total reward for task assignment A
$\mathbb{A}$	A spatial task assignment set

required to finish a task to denote $s.wl$ in our work). Next, $s.maxR$ is the maximum reward that the task requester can provide, and $s.pr$ is a penalty rate, which establishes a correlation between the completion time and the actual reward.

It is practical and reasonable to use the time required to finish a task s to denote $s.wl$ since the required time is often directly proportional to the workload of the task. It means that a task with more workload needs more time to be finished. The task workload depends on tasks. For example, a task of distributing leaflets includes 500 leaflets to be distributed. We can say that the workload of this task is 500 leaflets to be distributed, and distributing all the leaflets requires about one hour (that can be estimated by the spatial crowdsourcing platform or specified by the task requester). In this case, we can also use the number of leaflets to be distributed to denote the workload and estimate the time of distributing these leaflets. Thus, our algorithms are still applicable in such a case.

Definition 2 (*Worker*) A worker, denoted as $w = (l, on, sp, r)$, is a person who is willing to perform spatial tasks only if the worker is paid. A worker can be in an either online or offline mode. A worker is online when being ready to accept tasks and the worker is offline when being unavailable to perform tasks. An online worker is associated with a current location $w.l$, a recent online time $w.on$, a speed $w.sp$, and a reachable circular range with $w.l$ as the center and $w.r$ as the radius, in which w can accept assignments of tasks.

A worker is able to handle only one task at a time, which is reasonable in practice. A worker can be assigned a task only when being online and not performing any tasks. Once a task is assigned to a worker, the worker is considered as being offline until the task is completed.

Due to the constraint of workers' reachable ranges and tasks' expiration times, each task can be completed only by a subset of workers, called the *available worker set*.

Definition 3 (*Available Worker Set*) The available worker set for a task s, denoted as $AW(s)$, is a set of workers that satisfy the following two conditions: $\forall w \in AW(s)$,

1) worker w is able to arrive at the location of task s before its deadline, i.e., $t_{now} + t(w.l, s.l) < s.d$, and
2) task s is located in the reachable range of worker w, i.e., $d(w.l, s.l) \leq w.r$, where t_{now} is the current time, $t(a, b) = w.sp * d(a, b)$ is the travel time from location a to location b, and $d(a, b)$ is the travel distance from location a to location b. The above two conditions guarantee that workers in an available worker set can travel from their origins to the location of their reachable task s directly before it expires. If worker w is available for task s, i.e., $w \in AW(s)$, we say s is a reachable task of w and denote the reachable task set of w as $RS(w)$.

Definition 4 (*Worker Coalition*) Given a task s to be assigned and its available worker set $AW(s)$, the worker coalition for task s, denoted as $WC(s)$, is a subset of $AW(s)$ such that all the workers in $WC(s)$ have enough time to complete task s together before it expires, i.e., $\sum_{w \in WC(s)}(s.d - (t_{now} + t(w.l, s.l))) \geq s.wl$.

Taking Fig. 1 as a use case, the available worker set of task s_1 is $\{w_1, w_2, w_4, w_5\}$, where all the available workers can arrive at $s_1.l$ before $s_1.d$, and s_1 is reachable for them. $\{w_2\}$, $\{w_1, w_2\}$, and $\{w_1, w_2, w_5\}$ are worker coalitions for task s_1 since all the workers in each coalition can cooperate together to finish task s_1 before $s_1.d$.

As for task reward, taking longer time for completing a task (including waiting time for its assignment and task

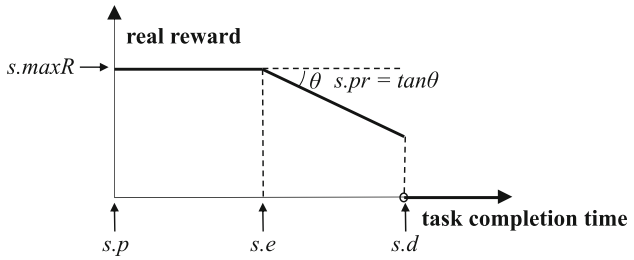


Fig. 2 Task reward pricing model

duration, i.e., from a task's publication time to its completion time) increases the probability of task failure in an SC environment, and thus reduces the rewards for workers. Considering the constraints on the tasks' expected completion time, deadline, required workload, and budget (i.e., the maximum reward the task requester can offer), we adopt the Reward Pricing Model (RPM) [34], which can effectively quantify the temporal constraints of tasks and is an important incentive mechanism to motivate workers to finish the assigned tasks on time. Specifically, RPM takes a single task s and one of its worker coalition into account, focusing on the task completion time and real reward (i.e., the requester's actual payment for the task), as depicted in Fig. 2.

With the main temporal constraints of the task (i.e., task's publication time $s.p$, expected completion time $s.e$, and deadline $s.d$), penalty rate $s.pr$, and maximum reward $s.maxR$, the RPM can be expressed as a formula shown as follows:

$$R_{WC(s)} = \begin{cases} s.maxR & s.p \le s.t_e \le s.e \\ s.maxR - s.pr * (s.t_e - s.e) & s.e < s.t_e \le s.d \\ 0 & s.t_e > s.d, \end{cases} \tag{1}$$

where $R_{WC(s)}$ represents the actual reward of task s when being completed by workers in $WC(s)$, and $s.t_e$ denotes the completion time of s given the worker coalition $WC(s)$. From Eq. 1 we can see, if a task can be completed before its expected completion time, workers will obtain the maximum reward. Without loss of generality, Eq. 1 models the penalty rate linearly when the task cannot be finished before its expected completion time but can be finished before its deadline. For example, in a house removal scenario, the task requester is happy to pay the maximum rewards to workers if the job can be finished before 17:00. After 17:00, the task requester may be a little bit disappointed but still would like to pay the reduced reward based on a penalty rate. However, this task requester will be too unsatisfied to pay any reward if the task cannot be completed by the hard deadline, e.g., by midnight today.

To compute the completion time $s.t_e$ of task s, we denote $s.t_s$ as the start time (i.e., time of assignment) of s, $T_{WC(s)}$ as the task duration of s (i.e., elapsed time from assignment to completion), and $w.wl(WC(s)) > 0$ as the workload contribution (measured by time) of w when task s is performed by $WC(s)$. From Fig. 3, which illustrates the workload allocation of worker coalition $\{w_1, w_2\}$ for task s_1 (where workers and tasks are from the running example in Fig. 1), it is easily understandable that, $\forall w \in WC(s)$, the task duration is equal to w's travel time plus the workload contribution, i.e.,

$$\begin{aligned} &T_{WC(s)} = t(w.l, s.l) + w.wl(WC(s)), \\ &\forall w \in WC(s), w.wl(WC(s)) > 0. \end{aligned} \tag{2}$$

By summing up the right side over all workers in coalition $WC(s)$, we have

$$T_{WC(s)} = \frac{\sum_{w \in WC(s)} t(w.l, s.l) + \sum_{w \in WC(s)} w.wl(WC(s))}{|WC(s)|}. \tag{3}$$

Given the fact that $s.wl = \sum_{w \in WC(s)} w.wl(WC(s))$, it comes to

$$T_{WC(s)} = \frac{\sum_{w \in WC(s)} t(w.l, s.l) + s.wl}{|WC(s)|}. \tag{4}$$

Finally, $s.t_e$ can be calculated as $s.t_e = s.t_s + T_{WC(s)}$, and each worker's workload can be calculated as $w.wl(WC(s)) = T_{WC(s)} - t(w.l, s.l)$. From the perspective of worker coalition, we assume that the goal of a worker joining a coalition is to increase the total reward of the coalition, which will accordingly lead to a satisfying reward for this worker.

Since we require that $w.wl(WC(s)) > 0$, if a worker's travel time exceeds the task duration, i.e., $t(w.l, s.l) \ge T_{WC(s)}$, then the worker w has no contribution to task s and should be removed from $WC(s)$. Additionally, as shown in the RPM in Fig. 2, when a worker coalition $WC(s)$ can cooperate to finish a task s before its expected completion time $s.e$, which means they obtain the maximum reward ($s.maxR$) of this task, adding more workers into $WC(s)$ cannot lead to a higher reward. It means that more workers in $WC(s)$ might not lead to earlier completion time or a higher total reward. This observation motivates the notion of minimal worker coalition.

Definition 5 (*Minimal Worker Coalition*) A worker coalition $WC(s)$ for task s is minimal (denoted by $MWC(s)$) if none of its subsets can obtain a reward that is equal to $R_{WC(s)}$.

In Fig. 1, although $\{w_2\}$, $\{w_1, w_2\}$ and $\{w_1, w_2, w_5\}$ are all worker coalitions for task s_1, $\{w_1, w_2, w_5\}$ is not a minimal worker coalition since $\{w_1, w_2\}$ can generate the same reward with $\{w_1, w_2, w_5\}$, i.e., $R_{\{w_1,w_2\}} = R_{\{w_1,w_2,w_5\}}$. For task s_4, both $\{w_6\}$ and $\{w_0, w_6\}$ are its worker coalitions, but

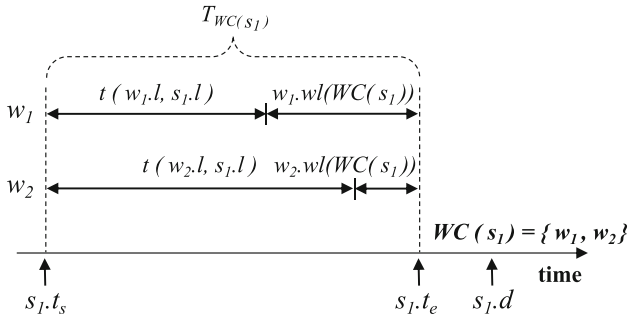


Fig. 3 Workload allocation of worker coalition $\{w_1, w_2\}$ for Task s_1

only $\{w_6\}$ is its minimal worker coalition. This is because w_6 can obtain the maximum reward of s_4 when conducting s_4 without the need of others' collaboration.

Definition 6 (*Spatial Task Assignment*) Given a set of workers W and a set of tasks S, a spatial task assignment, denoted by A, consists of a set of $(task, MWC)$ pairs in the form of $(s_1, MWC(s_1))$, $(s_2, MWC(s_2))$, ..., $(s_{|S|}, MWC(s_{|S|}))$, where $MWC(s_1) \bigcap MWC(s_2) \bigcap ... \bigcap MWC(s_{|S|}) = \emptyset$.

Let $A.R$ denote the total reward for task assignment A, i.e., $A.R = \sum_{s \in S, (s, MWC(s)) \in A} R_{MWC(s)}$ (where $R_{MWC(s)}$ can be calculated by Eq. 1), and $\mathbb{A}$ denote all the possible ways of assignments.

Problem Statement: Given a set of online workers W and a set of tasks S at a time instance, the CTA problem aims to find the global optimal assignment A_{opt}, such that the total reward can be maximized, i.e., $\forall A_i \in \mathbb{A}$, $A_i.R \leq A_{opt}.R$.

Lemma 1 *The CTA problem is NP-hard.*

Proof The lemma can be proved through a reduction from the 0-1 knapsack problem, which can be described as follows: given a set C with n items, in which each item $c_i \in C$ is labeled with a weight m_i and a value v_i, the 0-1 knapsack problem is to identify a subset C' of C that maximizes $\sum_{c_i \in C'} v_i$ subjected to $\sum_{c_i \in C'} m_i \leq M$, where M is a maximum weight capacity.

Considering the following instance of the CTA problem. We are given a task set S with n tasks, in which each task $s_i \in S$ is associated with the publication time $s_i.p = 0$, the expected completion time $s_i.e = 1$, the deadline $s_i.d = 1$, the workload $s_i.wl = m_i$ (corresponding to the weight m_i of the 0-1 knapsack problem), and the maximum reward $s_i.maxR = v_i$ (corresponding to the value v_i of the 0-1 knapsack problem). Additionally, we are given a worker set W with M workers, where each worker w_i is allowed to complete one workload only. All the workers and the tasks are in the same location. Therefore, the CTA problem is to identify a task subset S' of S that maximizes $\sum_{s_i \in S'} v_i$ subjected to $\sum_{s_i \in S'} m_i \leq M$.

If we can solve the CTA problem instance efficiently (i.e., in polynominal time), we can solve a 0-1 knapsack problem by transforming it to the corresponding CTA problem instance and then solve it efficiently. This contradicts the fact that the 0-1 knapsack problem is NP-hard [31], and so there cannot be an efficient solution to the CTA problem instance that is then NP-hard. Since the CTA problem instance is NP-hard, the CTA problem is also NP-hard.

3 Greedy approach

In this section, we design a greedy approach, which encourages worker coalitions to obtain more reward. This approach is based on the consensus that the nearby workers selected to perform a task can generate higher reward since the reward is non-increasing over time (i.e., it keeps stable with the maximum reward offered by the task requester at first and then gets lower, as shown in the Reward Pricing Model in Sect. 2), and the nearby workers are able to deal with more workloads to obtain more reward. In the following, we first introduce the concept of acceptance probability, based on which the greedy approach and its complexity analysis are given.

Acceptance Probability. Considering that the time utilization ratio (measured by workers' workload and travel time) of workers and rate of return (measured by the actual reward and maximal reward) of tasks, we introduce an acceptance probability to find the high-value task assignments (i.e., the task assignments ensuring high time utilization ratio of workers and high rate of return of tasks). Specifically, after assigning a worker coalition $MWC(s)$ to task s, we can calculate the acceptance probability $AP_{MWC(s)}$ (that means the probability that a task assignment $(s, MWC(s))$ is accepted) by Eq. 5.

$$AP_{MWC(s)} = \alpha \frac{\sum_{w \in MWC(s)} w.wl(MWC(s))}{\sum_{w \in MWC(s)} \left(t(w.l, s.l) + w.wl(MWC(s))\right)} + (1 - \alpha) \frac{R_{MWC(s)}}{s.maxR}, \quad (5)$$

where α is a parameter controlling the contribution of the time utilization ratio (i.e., $\frac{\sum_{w \in MWC(s)} w.wl(MWC(s))}{\sum_{w \in MWC(s)} \left(t(w.l, s.l) + w.wl(MWC(s))\right)}$) of workers and the rate of return (i.e., $\frac{R_{MWC(s)}}{s.maxR}$) of the task, and $0 \leq \alpha \leq 1$. Next, $w.wl(MWC(s))$ denotes the workload contribution (measured by time) of w when task s is performed by $MWC(s)$, and $t(w.l, s.l)$ denotes the travel time from location $w.l$ to location $s.l$. Further, $R_{MWC(s)}$ is the actual reward that $MWC(s)$ can obtain by performing task s, and $s.maxR$ is the maximal reward of s. Note that the acceptance probabilities among different tasks and worker coalitions are mutually independent, i.e., they are independently distributed.

Lemma 2 $0 \leq AP_{MWC(s)} \leq 1$.

Proof Considering that $w.wl(MWC(s)) \geq 0, t(w.l, s.l) \geq 0$, $w.wl(MWC(s)) \geq 0$, and $\alpha \geq 0$, we have:

$$\begin{aligned} 0 &\leq \alpha \frac{\sum_{w \in MWC(s)} w.wl(MWC(s))}{\sum_{w \in MWC(s)} \left(t(w.l, s.l) + w.wl(MWC(s))\right)} \\ &= \alpha \frac{\sum_{w \in MWC(s)} w.wl(MWC(s))}{\sum_{w \in MWC(s)} t(w.l, s.l) + \sum_{w \in MWC(s)} w.wl(MWC(s))} \\ &\leq \alpha \frac{\sum_{w \in MWC(s)} t(w.l, s.l) + \sum_{w \in MWC(s)} w.wl(MWC(s))}{\sum_{w \in MWC(s)} t(w.l, s.l) + \sum_{w \in MWC(s)} w.wl(MWC(s))} \\ &\leq \alpha \end{aligned} \tag{6}$$

Similarly, since $R_{MWC(s)} \geq 0$, $s.maxR \geq 0$, $1 - \alpha \geq 0$, and the actual reward $R_{MWC(s)}$ obtained by workers is not more than the maximal reward $s.maxR$ of s, the following inequality always holds:

$$0 \leq (1 - \alpha) \frac{R_{MWC(s)}}{s.maxR} \leq 1 - \alpha \tag{7}$$

By summing up Eqs. 6 and 7, we have $0 \leq AP_{MWC(s)} \leq 1$.

The acceptance probability $AP_{MWC(s)}$ is a probability ranging from 0 to 1, which considers the time utilization ratio of workers in $MWC(s)$ and the rate of return of task s. Apparently, the higher the time utilization ratio and the rate of return are, the more reward workers can obtain. Therefore, a higher acceptance probability means that workers in $MWC(s)$ can obtain more reward, which indicates that the task assignment $(s, MWC(s))$ is more likely to be accepted. Given the fact that $s.wl = \sum_{w \in WC(s)} w.wl(WC(s))$ and for convenience of calculation, Eq. 5 can be represented in the following:

$$AP_{MWC(s)} = \alpha \frac{s.wl}{|MWC(s)| T_{MWC(s)}} + (1 - \alpha) \frac{R_{MWC(s)}}{s.maxR}, \tag{8}$$

where $|MWC(s)|$ denotes the number of worker coalition $MWC(s)$, and $T_{MWC(s)}$ is the task duration of s (i.e., elapsed time from assignment to completion).

The Process of Greedy Approach. Algorithm 1 outlines the major procedure of the greedy approach, which takes a worker set W, a task set S, and an acceptance threshold η as input and outputs a task assignment result A. The algorithm starts with the calculation of the available worker set $AW(s)$ for each task s (line 3). After initialization of the current worker coalition (i.e., $WC(s) \leftarrow \emptyset$), the corresponding reward obtained by $WC(s)$ (i.e., $R_{WC(s)} \leftarrow 0$, $R_{WC(s)}$ is also the total reward of s), and a temporary variable (i.e., $R' \leftarrow 0$, line 4), for each task $s \in S$, the algorithm generates the minimal worker coalition $MWC(s)$ by choosing the closest workers who can contribute a higher overall reward

Algorithm 1: Greedy Approach

Input: Worker set W, task set S, acceptance threshold η
Output: Task assignment: A

```
1  A ← ∅;
2  for each s ∈ S do
3      Obtain the available worker set AW(s) from W;
4      WC(s) ← ∅; R_WC(s) ← 0; R' ← 0;
5      for the nearest worker w ∈ AW(s) do
6          R' ← R_WC(s)∪{w};
7          /*R_WC(s)∪{w} is computed based on Eq. 1.*/
8          AW(s) ← AW(s) − {w};
9          if R' = 0 and AW(s) = ∅ then
10             break;
11         if R' = 0 and AW(s) ≠ ∅ then
12             WC(s) ← WC(s) ∪ {w};
13         if R' > R_WC(s) then
14             WC(s) ← WC(s) ∪ {w};
15             R_WC(s) ← R';
16         else
17             MWC(s) ← WC(s);
18             R_MWC(s) ← R_WC(s);
19             A ← A ∪ (s, MWC(s));
20             break;
21     Calculate the acceptance probability AP_MWC(s) based on
       Eq. 8;
22     if AP_MWC(s) < η then
23         A ← A − (s, MWC(s));
24         S ← S − {s};
25     else
26         W ← W − MWC(s);
27         S ← S − {s};
28 return A;
```

for s and assigns the worker coalition $MWC(s)$ to s (lines 5–20). Specifically, by adding the closest worker $w \in AW(s)$ into the current worker coalition $WC(s)$, we can compute the reward obtained by coalition $WC(s) \cup \{w\}$ based on Eq. 1, i.e., $R_{WC(s)\cup\{w\}}$ (line 6). Then, we judge whether adding worker w can increase the total actual reward of s by performing the following actions:

1) if task s cannot be completed by workers in coalition $WC(s) \cup \{w\}$ (i.e., $R' = 0$), and there are no available workers (i.e., $AW(s) = \emptyset$), task s cannot be assigned to a suitable coalition (lines 9–10);
2) if task s cannot be completed by workers in coalition $WC(s) \cup \{w\}$ (i.e., $R' = 0$), but there are enough available workers (i.e., $AW(s) \neq \emptyset$), we add worker w into the current worker coalition $WC(s)$ (lines 11–12);
3) if adding worker w into $WC(s)$ can increase the reward obtained by $WC(s)$ (i.e., $R' > R_{WC(s)}$), worker w can be added into $WC(s)$, and the reward obtained by $WC(s)$ can be accordingly updated, i.e., $R_{WC(s)} \leftarrow R'$ (lines 13–15);
4) otherwise (i.e., when $R' \neq 0$ and adding worker w into $WC(s)$ cannot increase the reward obtained by $WC(s)$),

we can obtain the minimal worker coalition $MWC(s)$, the corresponding reward $R_{MWC(s)}$ for s, and the updated task assignment, i.e., $A \leftarrow A \cup (s, MWC(s))$ (lines 16–20). Next, we calculate the acceptance probability $AP_{MWC(s)}$ based on Eq. 8 (line 21). In case that the acceptance probability of task assignment $(s, MWC(s))$ is less than a given threshold η $(0 \leq \eta \leq 1)$, i.e., $AP_{MWC(s)} < \eta$, Algorithm 1 will quit performing task s, which means task s fails to be assigned (lines 22–24). Further, η can be specified by task requesters or SC platforms. Otherwise (i.e., $AP_{MWC(s)} \geq \eta$), the task assignment $(s, MWC(s))$ is regarded as high-value, and workers in $MWC(s)$ are assigned to perform task s. As a result, workers in $MWC(s)$ and task s can be removed from the worker set W to be assigned and task set S to be assigned (lines 25–27). Finally, Algorithm 1 will obtain a suitable task assignment result (line 28). For the example in Fig. 1, the greedy algorithm will yield a task assignment, $\{(s_1, \{w_1, w_2, w_4\}), (s_3, \{w_0, w_3\}), (s_4, \{w_6\})\}$, with the reward of 8.53 (that is 69.58% of the maximal reward), in which we set α as 0.5 and η as 0.4.

Time Complexity. It is easy to see the time complexity of Algorithm 1 is $O(|S| \cdot |W| \cdot |maxAW|)$, where $|S|$ is the number of tasks, $|W|$ is the number of workers, and $|maxAW|$ is the maximum number of available workers among all the tasks, i.e., $|maxAW| = \max_{s \in S} |AW(s)|$.

4 Equilibrium-based approach

Although the greedy algorithm can find a task assignment efficiently, it cannot guarantee the stability of the formed worker coalitions. The fundamental nature of the CTA problem is that each worker needs to choose a task to conduct by interacting with other workers during the task assignment, suggesting that the task selection for a worker depends on the decisions taken by the other workers. Such interdependent decisions can be modeled by game theory, where workers can be treated as independent players involved in a game. The proposed CTA system aims to assign a worker coalition to each task, so this is a process of coalition formation. Such a process can be considered to be a non-cooperative multi-player game, where each worker is a player that aims to find a suitable coalition (i.e., a suitable strategy) to achieve the optimal utility conditioned on the strategies of the other players in a myopic manner. Coalition formation is a non-cooperative game since it studies and models conflict situations among workers (players); that is, it studies situations where the utility of each worker depends not only on the strategy of the worker but also on the strategies of the other workers. Each worker will try to maximize the utility given the strategies of the other workers, and the outcome of the game depends on the strategies of all the workers. We aim to achieve stable coalitions so that no worker can improve their utility by a unilateral change to a different coalition if other workers stay in their coalitions. We adopt a best-response method to reach a pure Nash equilibrium within such a non-cooperative multi-player game.

To be more specific, our problem can be modeled as an exact potential game, which has at least one Nash equilibrium in pure strategy (a.k.a. pure Nash equilibrium) [22]. Then, we employ a best-response algorithm, one of the most basic tools in exact potential games as it is efficient in addressing the conflicts arising among players [6]. With the best-response dynamics, players are required to have their strategies updated sequentially and asynchronously on the basis of their best-response utility functions conditioned on the strategies of the other players in a myopic manner, which finally achieves a pure Nash equilibrium. A Nash equilibrium represents a state of the game where any single worker is incapable to improve their utility by making a unilateral shift from the assigned coalition to other coalitions when other workers stay in their assigned coalitions. This suggests that workers will voluntarily select the assigned tasks when they have freedom to do so. In such situation, the formed worker coalitions are regarded as stable coalitions. Nevertheless, this Nash equilibrium achieved by the best-response algorithm may be far from optimum as there can be many equilibrium points. In order to resolve this problem, we introduce a Simulated Annealing (SA) strategy into the best-response dynamics, which finds a better Nash equilibrium. With the help of the SA strategy, the updating process has a better chance to realize a better Nash equilibrium with higher total rewards. Finally, we analyze the feasibility of our solutions.

4.1 Game modeling and nash equilibrium

We first formulate our CTA problem as an n-player strategic game, $\mathcal{G} = (W, \mathbb{ST}, \mathbb{U})$, which is comprised of players, strategy spaces, and utility functions. It is specified as follows:

1) $W = \{w_1, ..., w_n\}$ $(n \geq 2)$ represents a finite set of workers playing the roles as the game players. In the rest of the paper, we will use player and worker interchangeably when the context is clear.
2) $\mathbb{ST} = \{ST_i\}_{i=1}^n$ is the overall strategy set of all the players, i.e., the strategy space of the game. ST_i is the finite set of strategies available to worker w_i, which contains w_i's reachable task set and null task (that means w_i does not choose any tasks to conduct), denoted as $ST_i = \{RS(w_i), null\}$ (where $RS(w_i)$ indicates the reachable task set of worker w_i and *null* represents the null task).
3) $\mathbb{U} = \{U_i\}_{i=1}^n$ denotes the utility functions of all the players, and $U_i : \mathbb{ST} \rightarrow \mathbb{R}$ is the utility function of player

w_i. For every joint strategy $\mathbf{st} \in \mathbb{ST}$, $U_i(\mathbf{st}) \in \mathbb{R}$ represents the utility of player w_i, which can be calculated as follows:

$$\begin{aligned} U_i(\mathbf{st}) =& R_{MWC(s)\cup\{w_i\}} - R_{MWC(s)} \\ & - (R_{MWC(s_0)} - R_{MWC(s_0)-\{w_i\}}), \end{aligned} \tag{9}$$

where $R_{MWC(s)\cup\{w_i\}}$ is the total reward obtained by coalition $MWC(s) \cup \{w_i\}$, $MWC(s) \cup \{w_i\}$ is the new worker coalition including $MWC(s)$ and $\{w_i\}$, $MWC(s_0) - \{w_i\}$ denotes the worker coalition (where w_i is removed from $MWC(s_0)$), s_0 denotes the task that is currently assigned to worker w_i, $MWC(s)$ and $MWC(s_0)$ are worker coalitions that w_i is willing to join and currently staying in, respectively. When the context of $\mathbf{st}$ is clear, we use U_i to denote $U_i(\mathbf{st})$.

In a strategic game, a policy profile $\pi^* = (\pi_1^*, ..., \pi_n^*)$ (where $\pi_i^* : ST_i \to [0, 1]$ is a probability distribution over ST_i) is called a Nash equilibrium with mixed strategies (a.k.a. mixed Nash equilibrium) if and only if for any $w_i \in W$, it holds that:

$$U_i\left(\pi^*\right) \geq \max_{\pi_i' \in \Sigma_i} U_i\left(\pi_1^*, \ldots, \pi_{i-1}^*, \pi_i', \pi_{i+1}^*, \ldots, \pi_n^*\right), \tag{10}$$

where Σ_i denotes the policy space of player w_i. For any given policy profile $\pi = (\pi_1, ..., \pi_n)$, $U_i(\pi) = \sum_{\mathbf{st}\in\mathbb{ST}} \pi_i(\mathbf{st}) U_i(\mathbf{st})$. The Nash equilibrium is a pure Nash equilibrium (i.e., Nash equilibrium with pure strategy) only when players play deterministic strategies, which means the probability of one strategy worker w_i can choose from ST_i is 1 while the rest strategies from ST_i are 0.

As proved by Nash et al. [24], a game with a finite number of players and a finite strategy set has a mixed Nash equilibrium, which only implies stable probability distributions over profiles rather than the fixed play of a particular joint strategy profile. This uncertainty is unacceptable in our scenario where each worker needs to have a definite strategy, i.e., selecting a task to conduct or doing nothing. Therefore, we show that our CTA game has pure Nash equilibriums, wherein each player can choose a strategy in a deterministic manner.

Given a joint strategy $\mathbf{st} = (st_1, ..., st_n) \in \mathbb{ST}$, st_i (i.e., $s \in RS(w_i)$ or *null*) represents the strategy chosen by player w_i ($0 < i \leq n$). As for player w_i, a joint strategy $\mathbf{st}_i \in \mathbb{ST}$ can also be denoted by $(st_i, \mathbf{st}_{-i})$, where $\mathbf{st}_{-i} = (st_1, ..., st_{i-1}, st_{i+1}, ..., st_n) \in \mathbb{ST}_{-i}$ is the joint strategies of all the other players.

Lemma 3 *The CTA game has pure Nash equilibriums.*

Proof To prove Lemma 3, we prove the CTA game is an Exact Potential Game (EPG) that has a global potential function onto which the incentive of all the players can be mapped. For an EPG that has at least one pure Nash equilibrium, the best-response framework always converges to a pure Nash equilibrium for countable strategies [22].

In the following part, we introduce the definition of EPG and show that the CTA game is an EPG.

Definition 7 (*Exact Potential Game*) A strategic game, $\mathcal{G} = (W, \mathbb{ST}, \mathbb{U})$, is an Exact Potential Game (EPG) if there exists a function, $\Phi : \mathbb{ST} \to \mathbb{R}$, such that for all $\mathbf{st}_i \in \mathbb{ST}$, it holds that,

$$\begin{aligned} &\forall w_i \in W \\ &\left(U_i(st_i', \mathbf{st}_{-i}) - U_i(st_i, \mathbf{st}_{-i}) = \Phi(st_i', \mathbf{st}_{-i}) - \Phi(st_i, \mathbf{st}_{-i})\right), \end{aligned} \tag{11}$$

where st_i' and st_i are the strategies that can be selected by worker w_i, $\mathbf{st}_{-i}$ is the joint strategy of the other workers except for worker w_i, and the function Φ is called an exact potential function for game $\mathcal{G}$.

Lemma 4 *CTA is an Exact Potential Game (EPG).*

Proof According to Definition 7, to prove that CTA is an exact potential game, we need to find an exact potential function satisfying Eq. 11. Therefore, we define a function, $\Phi(\mathbf{st}) = \sum_{s\in S} R_{MWC(s)}$, representing the total rewards for all tasks in S, where $R_{MWC(s)}$ is calculated by Eq. 1 that depends on the strategy st_i (i.e., the selected task) of each worker w_i in coalition $MWC(s)$. In other words, $R_{MWC(s)}$ depends on whether worker w_i selects task s or not. Then, it can be obtained that

$$\begin{aligned} &\Phi(st_i', \mathbf{st}_{-i}) - \Phi(st_i, \mathbf{st}_{-i}) \\ =& \left(R_{MWC(s_k)\cup\{w_i\}} + R_{MWC(s_g)} + \sum_{s\in S - s_k - s_g} R_{MWC}(s)\right) \\ & - \left(R_{MWC(s_k)} + R_{MWC(s_g)\cup\{w_i\}} + \sum_{s\in S - s_k - s_g} R_{MWC}(s)\right) \\ =& \left(R_{MWC(s_k)\cup\{w_i\}} - R_{MWC(s_k)}\right) \\ & - \left(R_{MWC(s_g)\cup\{w_i\}} - R_{MWC(s_g)}\right) \\ =& \Big(R_{MWC(s_k)\cup\{w_i\}} - R_{MWC(s_k)} \\ & - \left(R_{MWC(s_0)} - R_{MWC(s_0)-\{w_i\}}\right)\Big) \\ & - \Big(R_{MWC(s_g)\cup\{w_i\}} - R_{MWC(s_g)} \\ & - \left(R_{MWC(s_0)} - R_{MWC(s_0)-\{w_i\}}\right)\Big) \\ =& U_i(st_i', \mathbf{st}_{-i}) - U_i(st_i, \mathbf{st}_{-i}), \end{aligned} \tag{12}$$

where the tasks selected in strategies st_i' and st_i are s_k and s_g, respectively. In accordance with Definition 7, $\Phi(\mathbf{st})$ is an exact potential function satisfying Eq. 11, and the strategic

CTA game is an exact potential game. Since an EPG has at least one pure Nash equilibrium [22], the CTA game has pure Nash equilibriums.

Let st_i^* denote the best strategy that player w_i can make response to the strategy combination $\mathbf{st}_{-i}$ of others. Therefore, the utility $U_i(st_i^*, \mathbf{st}_{-i})$ is maximized for a given $\mathbf{st}_{-i}$. A pure Nash equilibrium is reached by the joint strategy $\mathbf{st}^* = (st_1^*, ..., st_n^*)$, as a result of which no player can have any gain in their utility by making change to their strategy unilaterally [23].

4.2 Best-response approach

As our CTA game has pure Nash equilibriums, we adopt a best-response approach to solve it, which generates a number of stable worker coalitions to perform the tasks by reaching a pure Nash equilibrium. Since it is impossible for a crowdsourced worker to know the strategy space of the other players and the resulting utility function, we assume that the Spatial Crowdsouring (SC) platform has access to such information to update workers' strategies (for selecting tasks) sequentially and asynchronously in order to reach a pure Nash equilibrium. This is a common assumption in SC studies [2, 32]. The benefit is that if all workers are in a Nash equilibrium, they are closely inter-connected, and the formed coalitions are stable, i.e., no worker can improve their utility by a unilateral change to a different coalition if other workers stay in their coalitions. The designed best-response algorithm consists of players taking turns to adapt their strategies (i.e., find their best-response strategies) based on the most recent known strategies of the others, which ends up reaching a Nash equilibrium that is a locally optimal task assignment. In the following, we first introduce the best-response strategy and then give the details of the best-response approach and its time complexity.

Best-response Strategy. For each worker $w_i \in W$, the best-response strategy is to find the best-response task to maximize the reward increase in the coalition that the worker is staying in. The best-response task s^* with the maximal reward increase can be calculated in Eq. 13.

$$\begin{aligned} s^* &= argmax_{s \in RS(w_i)} U_i(\mathbf{st}) \\ &= argmax_{s \in RS(w_i)} \big(R_{MWC(s) \cup \{w_i\}} - R_{MWC(s)} \\ &\quad - (R_{MWC(s_0)} - R_{MWC(s_0) - \{w_i\}})\big) \end{aligned} \tag{13}$$

The Process of Best-response Approach. A general framework of the best-response approach is illustrated in Algorithm 2.

Given a worker set W and a task set S to be assigned, the task assignment A is initialized as $\emptyset$ (line 1). The algorithm first randomly chooses an available worker for each task, obtains the corresponding strategy (i.e., a reachable task or doing nothing) for each worker, and updates the task assignment A accordingly (lines 2–9). Then, the algorithm iteratively adjusts each worker's strategy to the best-response strategy based on the current joint strategies of others until a Nash equilibrium (i.e., no one changes their strategies) is found (lines 11–30). At each iteration, only one worker is allowed to select the best-response strategy and the game is supposed to be played in sequence.

When there is no best-response task for worker w_i based on the current task assignment, w_i makes no change to the strategy (lines 15–16). For the worker selecting a best-response task, we check the current strategy of this worker as follows:

Algorithm 2: Best-response Approach

Input: Worker set W, task set S
Output: Task assignment: A

1 $A \leftarrow \emptyset$;
2 **for** *each task* $s \in S$ **do**
3 Obtain the available worker set $AW(s)$ from W and randomly assign an available worker, stored in $MWC(s)$, to s, where $\bigcap_{MWC(s)} = \emptyset$;
4 $A \leftarrow A \cup (s, MWC(s))$;
5 **for** *each worker* $w_i \in W$ **do**
6 **if** w_i *is assigned to a task* s **then**
7 $w_i.st \leftarrow s$;
8 **else**
9 $w_i.st \leftarrow null$;
10 $k \leftarrow 1$;
11 **repeat**
12 **for** *each worker* $w_i \in W$ **do**
13 find the best-response task s^* for w_i;
14 /*s^* can be obtained by Eq. 13*/
15 **if** $U_i(\mathbf{st}) \leq 0$ **then**
16 continue;
17 **else if** $w_i.st = null$ **then**
18 $w_i.st \leftarrow s^*$;
19 $MWC(s^*) \leftarrow MWC(s^*) \cup \{w_i\}$;
20 **else**
21 **if** *workers in* $\{MWC(w_i.st) - \{w_i\}\}$ *cannot complete task* $w_i.st$ *before its deadline* **then**
22 **for** *each worker* $w_j \in \{MWC(w_i.st) - \{w_i\}\}$ **do**
23 $w_j.st \leftarrow null$;
24 $MWC(w_i.st) \leftarrow \emptyset$;
25 **else**
26 $MWC(w_i.st) \leftarrow MWC(w_i.st) - \{w_i\}$;
27 $w_i.st \leftarrow s^*$;
28 $MWC(s^*) \leftarrow MWC(s^*) \cup \{w_i\}$;
29 $k \leftarrow k + 1$;
30 **until** $W.\mathbf{st}^k = W.\mathbf{st}^{k-1}$;
31 /*$W.\mathbf{st}^k$ denotes the strategies of all the workers in the kth iteration*/
32 update A;
33 **return** A;

1) in the event that the current strategy of the worker is doing nothing, i.e., $w_i.st = null$, we assign the best-response task to the worker (i.e., $w_i.st = s^*$) and update the minimal worker coalition for task s^* (lines 17–19);
2) in case that the current strategy of the worker involves a task (marked as $w_i.st$), which means w_i is assigned a task $w_i.st$ in coalition $MWC(w_i.st)$, the strategies of the other workers in coalition $MWC(w_i.st)$ are updated based on whether they are able to complete task $w_i.st$ together on time (lines 21–26). Subsequently, the strategy and worker coalition of w_i are updated (lines 27–28). Finally, we update the task assignment A according to the obtained Nash equilibrium (line 32).

Time Complexity. The time complexity of Algorithm 2 is $O(|S| \cdot |W| + |W| \cdot |maxRS| \cdot K)$, where $|S|$ is the number of tasks, $|W|$ is the number of workers, $|maxRS|$ is the maximum number of reachable tasks among all the workers (i.e., $|maxRS| = \max_{w \in W} |RS(w)|$), and K is the number of iterations to adjust each worker's best-response strategy until a Nash equilibrium is achieved.

4.3 Simulated annealing-based optimization strategy

Although the pure Nash equilibrium calculated by the best-response algorithm can generate an acceptable task assignment with stable worker coalitions, it is a local optima of the CTA problem and is not necessarily unique. Under the situations where multiple pure Nash equilibriums exist (i.e., the structure of the problem space is not smooth), it is desirable to obtain a better one than the one generated by the best-response algorithm. Simulated Annealing (SA) is a stochastic optimization procedure. It takes random walks through the problem space at successively lower temperatures, looking for points with better results (generated by the objective function) than the current local optimal point. Inspired by the success achieved by SA in solving discrete optimization problems [14], we employ it to search for better approximation to the global optimal task assignment.

In particular, when each worker updates the strategy st_i sequentially based on the given $\mathbf{st}_{-i}$ to maximize the utility function $U_i(st_i, \mathbf{st}_{-i})$, the workers may reach a Nash equilibrium that is a stable state. Considering that the search space is discrete (i.e., the strategy sets $\mathbb{ST} = \{ST_i\}_{i=1}^n$ are discrete), the Simulated Annealing (SA) [17] can be applied in the process of updating each worker's strategy in order for a better local optimum. SA is regarded as an efficient probabilistic scheme for the game updating to solve discrete optimization problems, evolving a discrete-time inhomogenous Markov chain, $x(k) = (st_1, ..., st_n)$. In our work, the state $x(k) = (st_1, ..., st_n)$ is the strategy combination of the workers at the kth iteration in Algorithm 2. For worker w_i, the strategy st_i can keep the current task s_0 or make change to one of the other reachable tasks (i.e., $RS(w_i) - \{s_0\}$). For simulation of the heat (randomness), it is assumed that worker w_i is able to change the current strategy at random by using one of the other reachable tasks with an identical probability $P_{st_i, st_i'} = 1/|RS(w_i)|$, where $st_i' = s$ $(s \in RS(w_i) - \{s_0\})$ or $st_i' = null$. Every single worker can update the strategy sequentially in line with the following rules.

1) If $U_i(st_i', \mathbf{st}_{-i}) \geq U_i(st_i, \mathbf{st}_{-i})$, then $x(k+1) = (st_i', \mathbf{st}_{-i})$.
2) If $U_i(st_i', \mathbf{st}_{-i}) < U_i(st_i, \mathbf{st}_{-i})$, then $x(k+1) = (st_i', \mathbf{st}_{-i})$ with probability

$$\begin{aligned} \mathcal{P} &= \exp\Big\{\frac{U_i(st_i', \mathbf{st}_{-i}) - U_i(st_i, \mathbf{st}_{-i})}{Tem(k)}\Big\}, \\ &= \exp\Big\{\frac{\Phi_i(st_i', \mathbf{st}_{-i}) - \Phi_i(st_i, \mathbf{st}_{-i})}{Tem(k)}\Big\}, \end{aligned} \tag{14}$$

where $Tem(k)(> 0)$ denotes the temperature at the kth iteration, which is in decline gradually throughout the updating process; otherwise, $x(k+1) = x(k) = (st_i, \mathbf{st}_{-i})$.

By adhering to the above rules, we can update lines 15–28 in Algorithm 2. The detailed pseudo-code is omitted due to space limit. Formally, the transition probability can be computed in Eq. 15.

$$\begin{aligned} &\mathbb{P}\left[x(k+1) = \left(st_i', \mathbf{st}_{-i}\right) | x(k) = (st_i, \mathbf{st}_{-i})\right] \\ &= \tfrac{1}{|RS(w_i)|} \exp\left\{\tfrac{\min(0, U_i(st_i', \mathbf{st}_{-i}) - U_i(st_i, \mathbf{st}_{-i}))}{Tem(k)}\right\} \end{aligned} \tag{15}$$

The function $Tem(k) : \mathbb{N} \to (0, \infty)$ is non-increasing, called cooling schedule, where $\mathbb{N}$ is the set of positive integers. From Eq. 15, it can be seen that the strategy selection is almost random when $Tem(k)$ is large while a better strategy with larger utility has a greater likelihood to be chosen when $Tem(k)$ approaches zero. According to the two strategy update rules of SA, we can see that although a worker updates the strategy with a certain probability when the utility of the update strategy is less than that of the current strategy, i.e., $U_i(st_i', \mathbf{st}_{-i}) < U_i(st_i, \mathbf{st}_{-i})$, the worker would definitely update the strategy when $U_i(st_i', \mathbf{st}_{-i}) \geq U_i(st_i, \mathbf{st}_{-i})$. As a whole, the rules are more likely to increase the total utility. Although allowing task selection with a smaller utility contributes to a decrease in the total utility, such "irregular" strategy selections have a potential to facilitate a better Nash equilibrium (i.e., a better task assignment), which is validated by the experiments in Sect. 6.

4.4 Convergence analysis

The question of convergence to a Nash equilibrium has attracted a great deal of attention in the game theory field

[10]. Therefore, we subsequently prove the convergence of our solution to a pure Nash equilibrium where no worker is incentivized to unilaterally deviate.

Lemma 5 *The best-response algorithm converges to a pure Nash equilibrium.*

Proof As depicted in Eq. 12, the utilities of all the workers are mapped onto a potential function (i.e., Φ), suggesting that the individually made adjustment to the strategy by each worker will result in a change to the utility of the worker and to the potential function with the same amount. For a potential game, each worker has the strategy updated sequentially for maximal utility by the best-response algorithm, and the potential function will reach a local maximum (i.e., a Nash equilibrium) accordingly, wherein the best-response dynamic is equivalent to a local search on the potential function of a potential game. The study [25] has proven that in any finite potential games, the sequential updates with best-response dynamic always converge to a Nash equilibrium.

Lemma 6 *The best-response algorithm with the simulated annealing optimization converges to a pure Nash equilibrium when the cooling schedule is regulated.*

Proof With the integration of the simulated annealing strategy into the best-response algorithm, randomness is added into the update process of workers' strategies, i.e., worker w_i can change the current strategy on a random basis by using one of the other reachable tasks with probabilities. Specifically, the process is "heated" up before "cooling" down, helping the potential function to avoid a local optimum (obtained by the best-response algorithm) and converge to another better Nash equilibrium, in which the cooling schedule ought to be regulated such that the process will eventually "freeze" (i.e., converge). Liu et al. [21] demonstrate that the convergence of simulated annealing strategy can be guaranteed when the cooling schedule is set as $Tem(k) = \frac{\beta}{log(k)}$ (where $\beta \geq D^*$ is a positive constant). If a joint strategy $\mathbf{st}$ has a path to the optimal joint strategy $\mathbf{st}^*$, D is defined as the depth such that the smallest value of the potential Φ along the path is $\Phi(\mathbf{st}) - D$. D^* denotes the maximum depth of the path starting from any joint strategy $\mathbf{st}$ and ending at the final joint strategy $\mathbf{st}^*$ if $\mathbf{st}$ has a path to $\mathbf{st}^*$.

The convergence of the best-response algorithm with simulated annealing is certain to proceed at a slower pace than that of the best-response algorithm. In Sect. 6, we provide experimental evidence for this statement and demonstrate a pure Nash equilibrium can effectively be calculated, as stated in our Lemma 5 and Lemma 6.

5 Priority-aware task assignment

Although stability is a desirable criterion for a worker coalition, a stable mechanism might not be attractive to workers because it might be unfair. For instance, if a worker contributes more than the other workers in a coalition and gets less reward, this worker probably loses faith in the mechanism. Moreover, a worker may have to wait for many assignment rounds without receiving any tasks if there are fewer tasks than workers. It is unfair since workers spending similar time on the SC platform receive inequitable rewards. Unfairly treated workers may reduce their working hours or leave the platform, eventually harming the platform.

Fairness is important in spatial crowdsourcing because of varying worker inputs, e.g., each worker's marginal contribution (the reward obtained by the worker when joining a coalition) and online time. To achieve a fair reward distribution among the members of a coalition, we introduce the concept of marginal contribution-based value and fairly compute each worker's reward in a coalition based on it. Then, we use a priority-aware fairness metric to assign tasks while taking workers' online time and travel time into account.

5.1 Marginal contribution-based reward distribution

Considering that the reward allocation for a worker is proportional to the worker's contribution to the coalition in a task assignment game, i.e., how much value the worker creates, we introduce the concept of Marginal Contribution-based Value. Specifically, The marginal contribution-based value of worker w_i in coalition $MWC(s)$ is the weighted average of all marginal contributions of w_i to all possible worker sequences from workers in $MWC(s)$. This is formalized as follows:

$$V(MWC(s), w_i) = \frac{\sum_{G \in \mathbb{G}(MWC(s))} \delta_i(G)}{|MWC(s)|!}, \tag{16}$$

$$\delta_i(G) = R_{G' \cup w_i} - R_{G'}, \tag{17}$$

where $V(MWC(s), w_i)$ is the marginal contribution-based value of worker w_i in worker coalition $MWC(s)$, G denotes a specific worker sequence based on the order of assignment, $\mathbb{G}(MWC(s))$ denotes all possible worker sequences, where workers come from $MWC(s)$, $\delta_i(G)$ is the marginal contribution of w_i by joining G, and $|MWC(s)|$ is the number of workers in coalition $MWC(s)$. The probability of each worker sequence occurring is the same and is equal to $\frac{1}{|MWC(s)|!}$. Further, $\delta_i(G)$ is defined in Eq. 17, where $R_{G' \cup w_i}$ denotes the reward obtained by workers in $G' \cup w_i$, and G' denotes the worker sequence before worker w_i joining G.

Taking task s_1 and worker coalition $\{w_1, w_2, w_4\}$ as an example, the calculation of the marginal contribution of w_i

Table 2 Marginal contribution calculation of w_1, w_2, and w_4 for completing s_1

Probability	Order of assignment	$\delta_1(G)$	$\delta_2(G)$	$\delta_4(G)$
1/6	w_1, w_2, w_4	$R_{\{w_1\}} = 0$	$R_{\{w_1,w_2\}} - R_{\{w_1\}} = 2.65$	$R_{\{w_1,w_2,w_4\}} - R_{\{w_1,w_2\}} = 0.12$
1/6	w_1, w_4, w_2	$R_{\{w_1\}} = 0$	$R_{\{w_1,w_4,w_2\}} - R_{\{w_1,w_4\}} = 0.12$	$R_{\{w_1,w_4\}} - R_{\{w_1\}} = 2.65$
1/6	w_2, w_1, w_4	$R_{\{w_2,w_1\}} - R_{\{w_2\}} = 0.15$	$R_{\{w_2\}} = 2.50$	$R_{\{w_2,w_1,w_4\}} - R_{\{w_2,w_1\}} = 0.12$
1/6	w_2, w_4, w_1	$R_{\{w_2,w_4,w_1\}} - R_{\{w_2,w_4\}} = 0.01$	$R_{\{w_2\}} = 2.50$	$R_{\{w_2,w_4\}} - R_{\{w_2\}} = 0.25$
1/6	w_4, w_1, w_2	$R_{\{w_4,w_1\}} - R_{\{w_4\}} = 0.15$	$R_{\{w_4,w_1,w_2\}} - R_{\{w_4,w_1\}} = 0.12$	$R_{\{w_4\}} = 2.50$
1/6	w_4, w_2, w_1	$R_{\{w_4,w_2,w_1\}} - R_{\{w_4,w_2\}} = 0.01$	$R_{\{w_4,w_2\}} - R_{\{w_4\}} = 0.25$	$R_{\{w_4\}} = 2.50$

($w_i \in \{w_1, w_2, w_4\}$), $\delta_i(G)$, is shown in Table 2, where $G \in \mathbb{G} = \{(w_1, w_2, w_4), (w_1, w_4, w_2), (w_2, w_1, w_4), (w_2, w_4, w_1), (w_4, w_1, w_2), (w_4, w_2, w_1)\}$. The marginal contribution-based value of each worker is computed as follows:

$$
\begin{aligned}
&V(\{w_1, w_2, w_4\}, w_1)\\
&= \frac{1}{6}(0 + 0 + 0.15 + 0.01 + 0.15 + 0.01) = 0.05,\\
&V(\{w_1, w_2, w_4\}, w_2)\\
&= \frac{1}{6}(2.65 + 0.12 + 2.50 + 2.50 + 0.12 + 0.25) = 1.36,\\
&V(\{w_1, w_2, w_4\}, w_4)\\
&= \frac{1}{6}(0.12 + 2.65 + 0.12 + 0.25 + 2.50 + 2.50) = 1.36
\end{aligned}
$$

The reward of a worker is calculated on the corresponding marginal contribution-based value, i.e., $R_{w_i}(MWC(s)) = V(MWC(s), w_i)$, where $R_{w_i}(MWC(s))$ denotes the reward obtained by worker w_i in coalition $MWC(s)$. The total reward obtained by coalition $\{w_1, w_2, w_4\}$ is 2.77. Workers w_1, w_2, and w_4 receive rewards 0.05, 1.36, and 1.36, respectively.

5.2 Priority-aware fairness

The CTA framework in the conference version [41] is designed assuming perfect rationality and self-interest among players. However, humans often care strongly about fairness [12]. Therefore, fairness should be taken into account in SC, where the allocation of tasks plays an important role. In SC, workers consider it fair that each worker gets a (slightly) different reward because they contribute differently, e.g., contribute different online time. For example, workers agree that a worker who has waited a long time for a task should have a higher priority to get a task than others. This additional information is denoted as the priority. Next, we proceed to adopt a priority-aware fairness criterion [13] that correlates with workers' reward, workload, and online time.

More formally, in an n-worker coalition $MWC(s)$, we assign a priority-aware utility (PAU), $PAU(w_i, s)$, to worker w_i to measure the priority-aware fairness, which is calculated using the following equations.

$$PAU(w_i, s) = \frac{\sum_{j=1, i\neq j}^{n} \zeta(i, j)}{n - 1}, \tag{18}$$

$$\zeta(i, j) = \begin{cases} 1 \exists \gamma \in [\gamma_{\min}, \gamma_{\max}] \\ \left(\frac{R_{w_i}(MWC(s))}{f_\gamma(i)} = \frac{R_{w_j}(MWC(s))}{f_\gamma(j)}\right) \\ 0 \qquad \text{otherwise,} \end{cases} \tag{19}$$

$$f_\gamma(i) = \frac{1}{n} + \gamma\left(p_i - \frac{1}{n}\right), \tag{20}$$

$$p_i = 1 - \frac{1}{t_{now} - w_i.on + 1}, \tag{21}$$

where $\zeta(i, j)$ denotes a scoring function for any pair of workers w_i and w_j. Function $\zeta(i, j)$ is calculated in Eq. 19, where $\gamma \in [\gamma_{\min}, \gamma_{\max}]$ is a greediness parameter, and $R_{w_i}(MWC(s))$ is the reward of w_i in coalition $MWC(s)$. To enable flexible fairness, we use $[\gamma_{\min}, \gamma_{\max}]$ to denote the upper and lower bounds of parameter γ in Eq. 19, thus making it possible to specify how tolerant we are with respect to differences in worker rewards. We say that workers w_i and w_j have a fair share with respect to each other if their rewards satisfy $\frac{R_{w_i}(MWC(s))}{f_\gamma(i)} = \frac{R_{w_j}(MWC(s))}{f_\gamma(j)}$ with $\gamma \in [\gamma_{\min}, \gamma_{\max}]$. Further, $f_\gamma(i)$ denotes a fairness function of worker w_i (cf. Eq. 20), and p_i denotes the priority value of worker w_i, which is measured according to the online duration of worker w_i (i.e., the difference between the current time t_{now} and w_i's online time $w_i.on$), cf. Eq. 21. A higher priority-aware utility implies a more fair task assignment.

5.3 Priority-aware task assignment approaches

The original CTA problem is converted into a priority-aware CTA problem by considering the priority-aware fairness of workers. The next problem is how to combine the priority-aware fairness with the existing objective of maximizing the total reward. In other words, the assignment should provide priority-aware fairness without sacrificing the total reward. To enable this, we improve the greedy and equilibrium-based task assignment approaches to maximize the total reward with priority-aware fairness as a constraint.

Algorithm 3: Priority-aware Greedy Approach

Input: Worker set W, task set S, acceptance threshold η, priority-aware threshold δ_g
Output: Task assignment: A

1 $A \leftarrow \emptyset$;
2 **for** *each* $s \in S$ **do**
3 Obtain the available worker set $AW(s)$ from W;
4 $WC(s) \leftarrow \emptyset$; $R_{WC(s)} \leftarrow 0$; $R' \leftarrow 0$;
5 **for** *the nearest worker* $w \in AW(s)$ **do**
6 $R' \leftarrow R_{WC(s)\cup\{w\}}$;
7 /*$R_{WC(s)\cup\{w\}}$ is computed based on Eq. 1.*/
8 $AW(s) \leftarrow AW(s) - \{w\}$;
9 Compute w's marginal contribution-based reward $R_w(WC(s) \cup \{w\})$ based on Eq. 16 and compute w's priority-aware utility $PAU(w, s)$ based on Eq. 18 accordingly;
10 **if** $PAU(w, s) > \delta_g$ **then**
11 **if** $R' = 0$ *and* $AW(s) = \emptyset$ **then**
12 break;
13 **if** $R' = 0$ *and* $AW(s) \neq \emptyset$ **then**
14 $WC(s) \leftarrow WC(s) \cup \{w\}$;
15 **if** $R' > R_{WC(s)}$ **then**
16 $WC(s) \leftarrow WC(s) \cup \{w\}$;
17 $R_{WC(s)} \leftarrow R'$;
18 **else**
19 $MWC(s) \leftarrow WC(s)$;
20 $R_{MWC(s)} \leftarrow R_{WC(s)}$;
21 $A \leftarrow A \cup < s, MWC(s) >$;
22 break;
23 Calculate the acceptance probability $AP_{MWC(s)}$ based on Eq. 8;
24 **if** $AP_{MWC(s)} < \eta$ **then**
25 $A \leftarrow A - < s, MWC(s) >$;
26 $S = S - \{s\}$;
27 **else**
28 $W = W - MWC(s)$;
29 $S = S - \{s\}$;
30 **return** A.

5.3.1 Priority-aware greedy approach

We improve the greedy task assignment approach by introducing a priority-aware threshold, δ_g, which can guarantee the priority-aware fairness among workers to some extent and be set by users or the SC platform.

The Process of Priority-aware Greedy Approach. The priority-aware greedy approach, shown in Algorithm 3, differs mainly from the original approach in Algorithm 1 in that when assigning nearby workers to a task (lines 2–29), the priority-aware utility of the worker is calculated (line 9), and a worker is allowed to join a coalition only if the priority-aware utility of the worker exceeds the user- or platform-specified threshold δ_g (line 10), which guarantees priority-aware fairness of the existing task assignment to some extent.

Time Complexity. The time complexity of Algorithm 3 is $O(|S| \cdot |W| \cdot |maxAW| \cdot |maxMWC|!)$, where $|S|$ is the number of tasks, $|W|$ is the number of workers, $|maxAW|$ is the maximum number of available workers among all the tasks, and $|maxMWC|$ is the maximum number of workers among all coalitions, i.e., $|maxMWC| = \max_{s\in S} |MWC(s)|$. In practice, the number $|MWC(s)|$ of workers in coalition $MWC(s)$ is small, which means that the time complexity is acceptable.

Algorithm 4: Priority-aware Equilibrium-based Approach

Input: Worker set W, task set S, priority-aware threshold δ_e
Output: Task assignment: A

1 $A \leftarrow \emptyset$;
2 **for** *each task* $s \in S$ **do**
3 Obtain the available worker set $AW(s)$ from W and randomly assign an available worker, stored in $MWC(s)$, to s, where $\bigcap_{MWC(s)} = \emptyset$;
4 $A \leftarrow A \cup (s, MWC(s))$;
5 **for** *each worker* $w_i \in W$ **do**
6 Obtain reachable task set $RS(w_i)$ based on $AW(s)$;
7 **if** w_i *is assigned to a task* s **then**
8 $w_i.st \leftarrow s$;
9 **else**
10 $w_i.st \leftarrow null$;
11 $k \leftarrow 1$;
12 **repeat**
13 **for** *each worker* $w_i \in W$ **do**
14 **for** *each task* $s \in RS(w_i)$ **do**
15 **if** s *is the best-response task for* w_i *and* $U_i(\mathbf{st}) > 0$ **then**
16 **if** $w_i.st = null$ **then**
17 $w_i.st \leftarrow s$;
18 $MWC(s) \leftarrow MWC(s) \cup \{w_i\}$;
19 **else if** *workers in* $\{MWC(w_i.st) - \{w_i\}\}$ *cannot complete* $w_i.st$ *before deadline* **then**
20 **for** *each worker* $w_j \in \{MWC(w_i.st) - \{w_i\}\}$ **do**
21 $w_j.st \leftarrow null$;
22 $MWC(w_i.st) \leftarrow \emptyset$;
23 **else**
24 $MWC(w_i.st) \leftarrow MWC(w_i.st) - \{w_i\}$;
25 $w_i.st \leftarrow s$;
26 $MWC(s) \leftarrow MWC(s) \cup \{w_i\}$;
27 **else**
28 Compute $R_{w_i}(MWC(s))$ and $PAU(w_i, s)$;
29 **if** $PAU(w_i, s) > \delta_e$ **then**
30 s is assigned to w_i with probability $\mathcal{P}$ (see Eq. 14);
31 $k \leftarrow k + 1$;
32 **until** $W.\mathbf{st}^k = W.\mathbf{st}^{k-1}$;
33 update A;
34 **return** A.

5.3.2 Priority-aware equilibrium-based approach

We formulate the priority-aware CTA problem as an n-player strategic game, $\mathcal{G} = (W, \mathbb{ST}, \mathbb{U})$, as in Sect. 4.1.

The Process of Priority-aware Equilibrium-based Approach. The improved priority-aware equilibrium-based approach is given in Algorithm 4, where the initialization is the same as in the original Equilibrium-based approach (Algorithm 2 in Sect. 4.2). Specifically, we first compute the available worker set and assign an available worker to each task randomly (lines 2–4). Then, the reachable task set can be computed for each worker (lines 5–6), and each worker updates their strategy (lines 7–10).

Next, we check each task s in worker w_i's reachable task set $RS(w)$ (lines 13–14). If s is the best-response task for w_i, we assign it to w_i (lines 15–26), which is the same as in Algorithm 2. Otherwise, we compute w_i's the marginal contribution-based reward $R_{w_i}(MWC(s))$ based on Eq. 16 and compute w_i's priority-aware utility $PAU(w_i, s)$ based on Eq. 18 (line 28). If the priority-aware utility of w_i exceeds a user- or platform-specified threshold (i.e., $PAU(w_i, s) > \delta_e$), s is assigned to w_i with probability $\mathcal{P}$, where $\mathcal{P}$ is calculated based on Eq. 14 (lines 29–13).

Time Complexity. The time complexity of Algorithm 4 is $O(|S| \cdot |W| + |W| \cdot |maxRS| \cdot K \cdot |maxMWC|!)$, where $|maxRS|$ is the maximum number of reachable tasks among all the workers (i.e., $|maxRS| = \max_{w \in W} |RS(w)|$), and K is the number of iterations to adjust each worker's best-response strategy until a Nash equilibrium is achieved.

6 Experimental study

We proceed to evaluate the performance of the proposed methods using both real and synthetic datasets. All experiments are performed on an Intel (R) Xeon (R) CPU E5-2650 v2 @ 2.60 GHz with 128 GB RAM.

6.1 Experimental setup

The experiments are carried out using two datasets, named *gMission* (GM) and *synthetic* (SYN). First, gMission is an open source SC dataset [1] in which each task is associated with a location, a deadline, and a reward (regarded as its maximal reward), and in which each worker is associated with a location and a reachable radius. The publication time of all tasks is set to 0, which is also the current time. Then, the online time of workers is generated uniformly from values in the range $[-5, 0]$ to guarantee that workers are online before or at the current time. As gMission is not associated with a workload, expected completion time, and penalty rate of tasks, we uniformly generate these from ranges $[\frac{2}{5} \cdot s.d, 2 \cdot s.d]$, $[\frac{2}{5} \cdot s.d, \frac{3}{5} \cdot s.d]$, and $[0, \frac{s.maxR}{s.d - s.e}]$ (to ensure that the actual reward obtained by workers is non-negative), respectively, where $s.d$ is the deadline of s and $s.maxR$ is the reward of each task s. It is common practice in experimental studies of SC platforms to generate the values of these attributes in a uniform manner [11, 27]. The speed of workers are set to 5km/h.

For the synthetic dataset, based on the observation from gMission that the locations of workers/tasks are uniformly distributed in space, we generate the locations of workers/tasks following a uniform distribution. The maximum reward of each task is set following a Gaussian distribution since it is influenced by complex variables in practice. The workload is uniformly generated from range [2, 10]. Other settings (i.e., the publication time, the penalty rate of each task, and the online time and speed of each worker) in the synthetic dataset are set to resemble the settings seen in gMission.

We study the performance of the following algorithms:

1) OTA: The Optimal Task Assignment algorithm based on tree decomposition. OTA first finds all minimal worker coalitions for each task by utilizing dynamic programming and then applies the tree-decomposition-based algorithm [42] to identify the optimal task assignment with the maximum reward.
2) GTA: The Greedy Task Assignment algorithm, where α is set to 0.5 and the acceptance threshold η is set to 0.4.
3) GTA+PAU: The PAU-based GTA, where the threshold on the PAU is set to 0.03 and $[\gamma_{\min}, \gamma_{\max}]$ is set to [0.3, 1.5].
4) BR: The Best-Response approach.
5) BR+SA: The Best-Response approach with Simulated Annealing optimization, where the cooling schedule is given by $Tem(k) = \frac{1}{log(k+1)}$, where k denotes the kth iteration of the algorithm.
6) BR+SA+PAU: The PAU-based BR+SA, where the threshold on the PAU is set to 0.25 and $[\gamma_{\min}, \gamma_{\max}]$ is set to [0.4, 0.6].

Three main metrics are used for comparing the algorithms: *Total reward*, *Average Payoff Difference*, and *CPU time* for finding the final task assignment. The *Average Payoff Difference*, P_{dif}, that measures the payoff difference among workers in a coalition by considering the rewards and the online times of workers, is calculated as follows:

$$\begin{aligned} P_{\text{dif}} &= \frac{1}{|S|} \sum_{s \in S} \Delta P_{\max}(WC(s)), \\ &\quad \Delta P_{\max}(WC(s)) \\ &= \max_{w_i, w_j \in WC(s)} \left\{ \left| \frac{R_{w_i}(WC(s))}{s.T - w_i.on} - \frac{R_{w_j}(WC(s))}{s.T - w_j.on} \right| \right\}, \end{aligned} \tag{22}$$

where $|S|$ is the number of tasks, $\Delta P_{\max}(WC(s))$ is the maximal payoff difference among workers in coalition $WC(s)$,

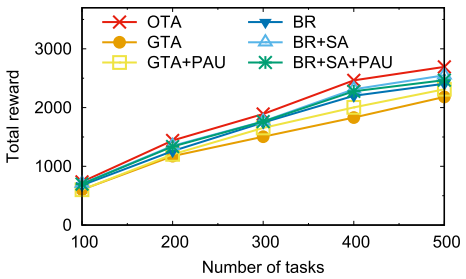
Table 3 Parameter settings

Parameter	Values
Number of tasks, $\|S\|$ (GM)	100, 200, 300, 400, 500
Number of tasks, $\|S\|$ (SYN)	1K, 2K, 3K, 4K, 5K
Number of workers, $\|W\|$ (GM)	100, 200, 300, 400, 500
Number of workers, $\|W\|$ (SYN)	1K, 2K, 3K, 4K, 5K
Reachable distance of workers, r (km) (SYN)	1, 2, 3, 4, 5
Expected completion time of tasks, e (SYN)	2 h, 4 h, 6 h, 8 h, 10 h
Time between the expected completion time and the deadline of tasks, $d - e$ (GM)	2 h, 4 h, 6 h, 8 h, 10 h

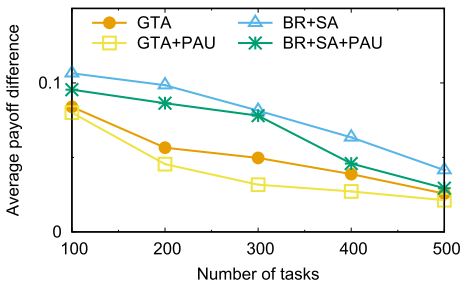
$\frac{R_{w_i}(WC(s))}{s.T - w_i.on}$ is the payoff obtained by worker w_i for finishing task s, $s.T$ is the completion time of task s, and $w_i.on$ is the online time of worker w_i. Apparently, it is more fair if the payoff difference among workers is smaller. In other words, a task assignment with a smaller average payoff difference is a more fair task assignment. Table 3 shows parameter settings, where default values are underlined.

6.2 Experimental results

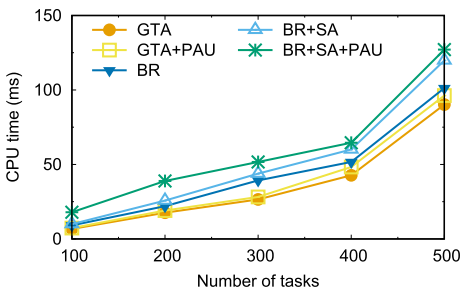
Effect of $|S|$. To study the scalability of all the algorithms, we generate five datasets containing 100 to 500 (1, 000 to 5, 000) tasks by random selection from the gMission dataset (synthetic dataset). As shown in Figs. 4a and 5a, the total reward of all the methods exhibits a similar increasing trend when $|S|$ grows. Since OTA is the optimal task assignment algorithm, it achieves the highest total reward, followed by BR+SA, BR+SA+PAU, BR, GTA+PAU, and GTA, on both gMission and the synthetic datasets. BR+SA can obtain at most 96% of the maximal reward, and its reward is consistently higher than that of BR (by up to 8%), which demonstrates the superiority of the simulated annealing optimization strategy. BR and GTA can achieve up to 93% and 82% of the optimal reward, respectively. Figures 4b and 5b show the experimental results about the average payoff difference, which aim to demonstrate the effectiveness of considering the priority-aware fairness (i.e., PAU) in task assignment. Therefore, we only compare the methods (i.e., GTA+PAU and BR+SA+PAU) that consider priority-aware fairness with their counterparts (i.e., GTA and BR+SA) in these experiments. We can see that although BR+SA obtains higher rewards than BR+SA+PAU (up to 2%), its average payoff difference is higher than that of BR+SA+PAU (up to 41%). The payoff difference of the algorithms decreases with increasing $|S|$ since workers have more choices when more tasks are to be assigned, which enables them to choose suitable tasks that lead to smaller payoff differences. It is noteworthy that GTA+PAU performs better than GTA in terms of total rewards and average payoff differences, as shown in Figs. 4a, b, and 5a, b. In Figs. 4c, d and 5c, d, despite



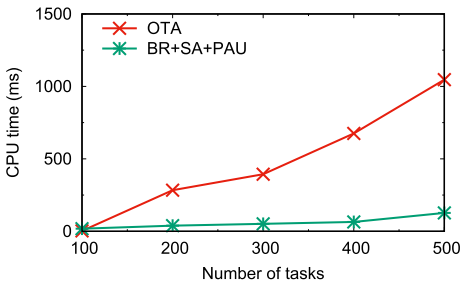
(a) Total Reward



(b) Average Payoff Difference

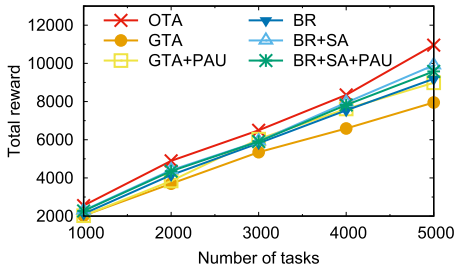


(c) CPU Time

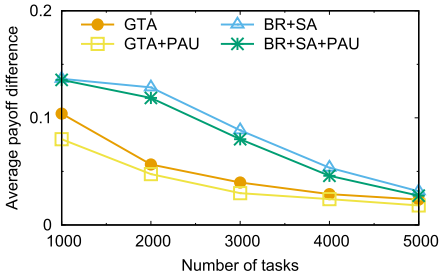


(d) CPU Time

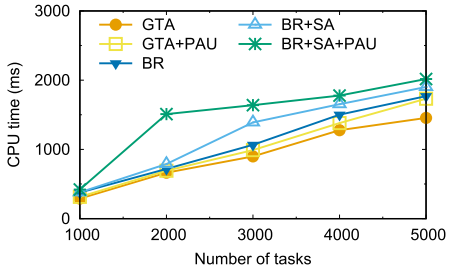
Fig. 4 Effect of $|S|$ on gMission



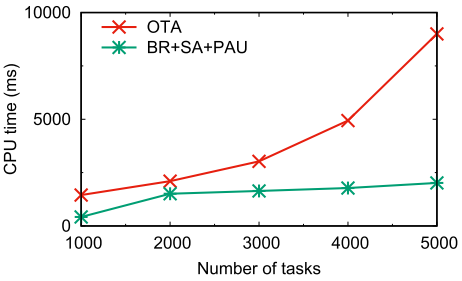
(a) Total Reward



(b) Average Payoff Difference

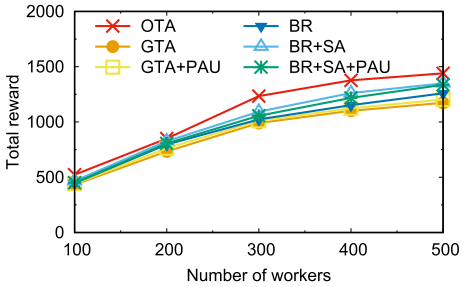


(c) CPU Time

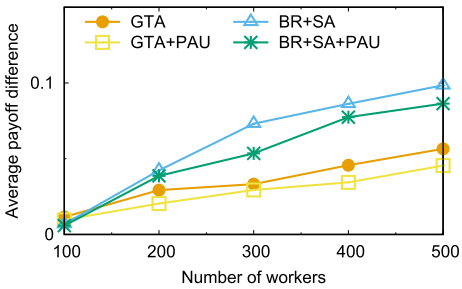


(d) CPU Time

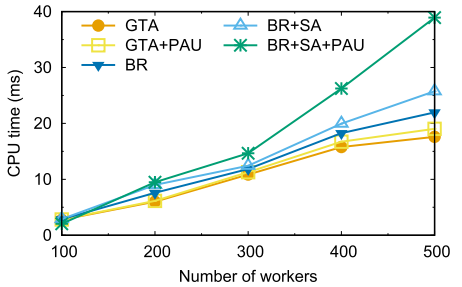
Fig. 5 Effect of $|S|$ on the Synthetic Dataset



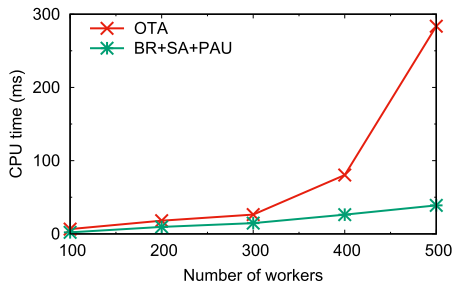
(a) Total Reward



(b) Average Payoff Difference



(c) CPU Time

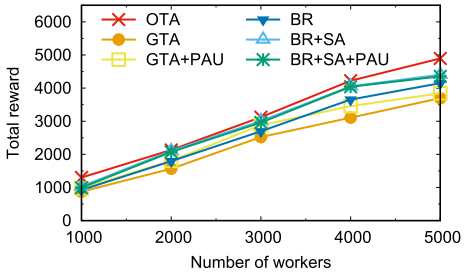


(d) CPU Time

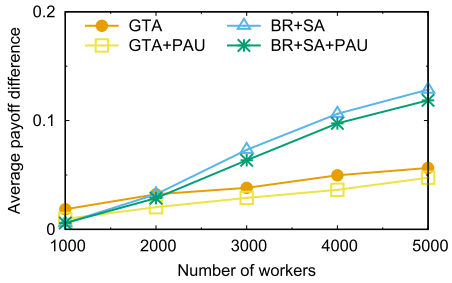
Fig. 6 Effect of $|W|$ on gMission

the CPU time of all methods increase as $|S|$ increases, our proposed algorithms (including GTA, GTA+PAU, BR, BR+SA, and BR+SA+PAU) deliver clearly superior performance to OTA. OTA deteriorates at a significantly faster pace in terms of efficiency. As expected, GTA is the fastest algorithm, which can improve efficiency by 11%–32% (23%–39%) compared with BR (BR+SA), while it generates smaller rewards when compared to the other algorithms. Moreover, independently of $|S|$, GTA and BR+SA always run faster than their counterparts (i.e., GTA+PAU and BR+SA+PAU) that take into account fairness. This is because GTA+PAU and BR+SA+PAU have to compute the PAU when assigning tasks.

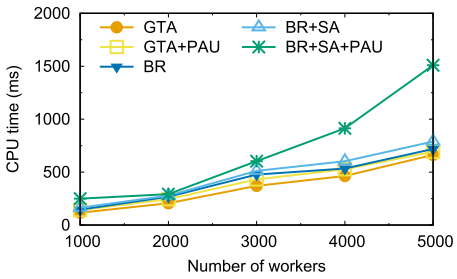
Effect of $|W|$. Next, we study the effect of $|W|$, the number of workers to be assigned. As shown in Figs. 6a, c, and 7a, c, BR, BR+SA, and BR+SA+PAU can achieve higher



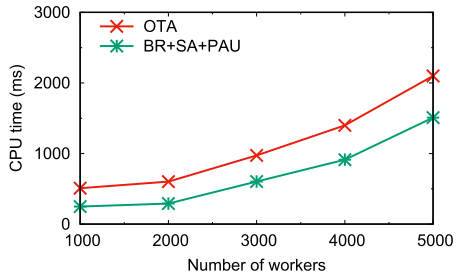
(a) Total Reward



(b) Average Payoff Difference

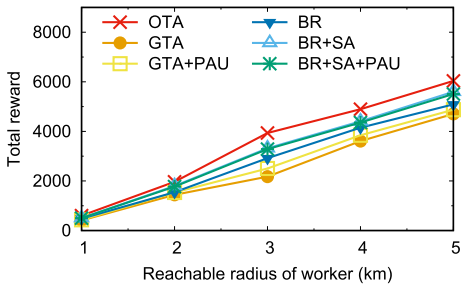


(c) CPU Time

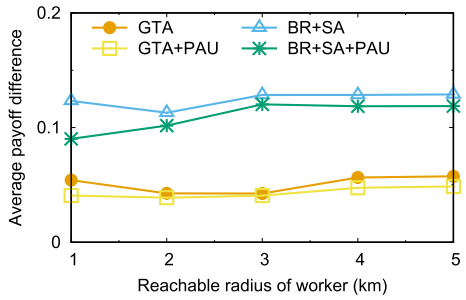


(d) CPU Time

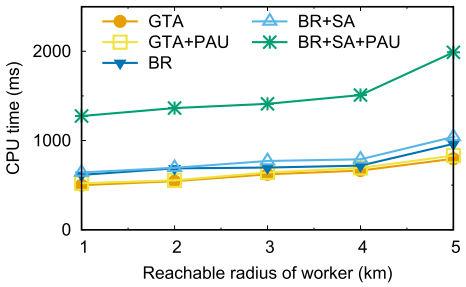
Fig. 7 Effect of $|W|$ on the synthetic dataset



(a) Total Reward



(b) Average Payoff Difference



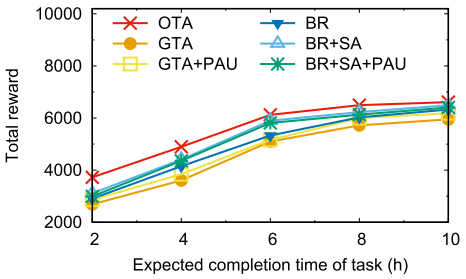
(c) CPU Time

Fig. 8 Effect of r on the synthetic dataset

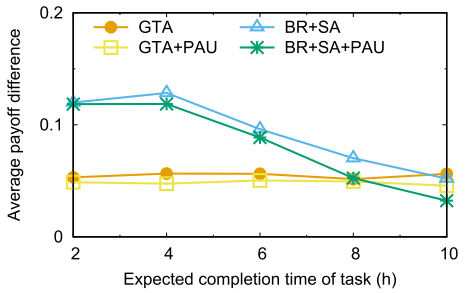
global rewards than GTA-related methods while sacrificing some efficiency. However, the computational efficiency of BR, BR+SA, and BR+SA+PAU are acceptable. Although the total reward of OTA is the highest, it is time-consuming compared with other methods, as shown in Figs. 6d and 7d. More specifically, BR+SA can achieve up to 98% of the maximal reward, and its CPU time is significantly less than that of OTA, i.e., the CPU time of BR+SA is only 10%–53% of OTA's. In contrast, the reward obtained by BR (GTA) is only 71%–93% (67%–86%) of the maximal reward. From Figs. 6b and 7b, we can see that the average payoff difference shows an upward trend when increasing $|W|$. This is because each task tends to be assigned to a coalition with more workers when the number of workers increases, which leads to higher payoff differences. GTA+PAU and BR+SA+PAU still outperform their own counterparts (i.e., GTA and BR+SA) in terms of fairness.

To save space, in the following experiments, we omit the CPU time of OTA that is excessive, and we omit results on gMission as these are similar to those of the synthetic dataset.

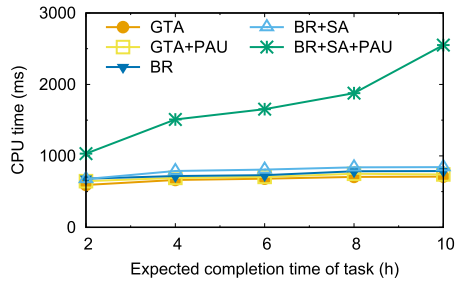
Effect of r. Fig. 8 shows the effect of workers' reachable distance, r, on the performance of all the algorithms. When the reachable distance is increased, workers can reach more tasks, enabling them to select tasks with higher rewards, which explains the increasing trends of the total rewards with growing r in Fig. 8a. When it comes to the average pay-



(a) Total Reward

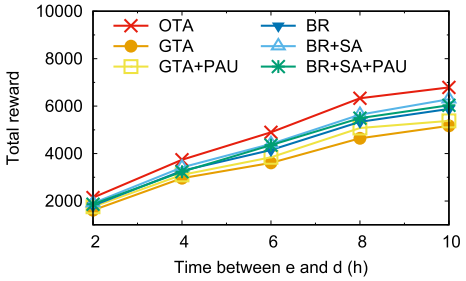


(b) Average Payoff Difference

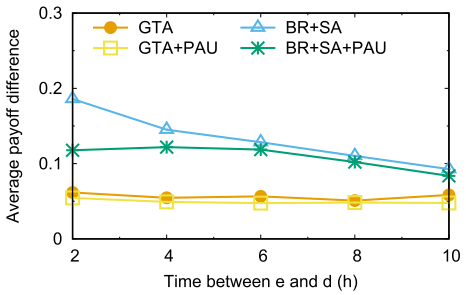


(c) CPU Time

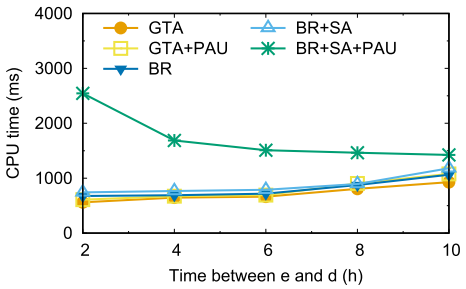
Fig. 9 Effect of e on the synthetic dataset



(a) Total Reward



(b) Average Payoff Difference



(c) CPU Time

Fig. 10 Effect of $d - e$ on the synthetic dataset

off difference, as expected, the PAU-related methods (i.e., GTA+PAU and BR+SA+PAU) show a clearly superior performance over the methods without PAU (i.e., GTA and BR+SA), as seen in Fig. 8b. Moreover, the CPU time of all the approaches increases with increasing r since workers consider more tasks when finding suitable tasks—cf. Fig. 8c.

Effect of e. Next, we study how the expected completion time of tasks affects the performance. Figure 9a shows that the total rewards of all methods increase gradually with increasing e since a larger e implies that more tasks can reach the maximal rewards. OTA still obtains the maximal rewards, and BR+SA outperforms BR and GTA. It is noticeable, however, that all the methods tend to maintain stability when $e > 8h$, which may be due to the fact that a majority of the tasks can be completed before $8h$ to achieve their own maximal rewards. In Fig. 9b, the average payoff difference of BR+SA and BR+SA+PAU decreases when $e > 4$, since it is more likely that workers can choose suitable tasks that lead to smaller payoff differences with larger e. In terms of CPU time in Fig. 9c, all methods except BR+SA+PAU exhibit slight ascending trends. The CPU time of BR+SA+PAU increases sharply when e gets larger. This is because with larger e, workers tend to form larger worker coalitions to achieve the maximum rewards for tasks, and computing the marginal contribution-based values for these coalitions takes more CPU time.

Effect of $d - e$. In the final set of experiments, we study the effect of $d - e$. Not surprisingly, as can be seen in Fig. 9a, all the approaches lead to higher rewards when deadlines are relaxed. A larger $d - e$ means that each worker on average has more reachable tasks, which increases the total rewards. Another observation is that the performance gap between the BR-related approaches and the GTA-related algorithms in terms of the total reward is also increasing. This is due to the fact that when applying the BR-related algorithms, the total reward is more sensitive to the average number of available worker sets for each task, which increases with $d - e$. In such

circumstances, the benefits of the BR-related approaches become more significant. In terms of the average payoff difference in Fig. 9b, we observe that the PAU-related methods consistently perform better than their own counterparts without considering priority-aware fairness. In addition, the average payoff difference of BR+SA and BR+SA+PAU show a decreasing trend when $d - e$ increases, for reasons similar to those explaining the effects of e, i.e., the larger $d - e$ is, the more chance the SC server has to assign workers tasks with smaller average differences. From Fig. 9c, we can see that all the approaches except BR+SA+PAU use more CPU time when $d - e$ increases. BR+SA+PAU costs less CPU time with increasing $d - e$. This is due to the fact that each task has more available workers when $d - e$ gets larger, for which the minimal worker coalitions for each task are more likely to be smaller. As a result, the efficiency of computing marginal contribution-based values is improved.

Summary: The take-away message of our empirical study can be summarized as follows:

1. OTA achieves the maximum rewards but sacrifices some efficiency, which can be applied in small SC applications, e.g., on-wheel meal-ordering service (e.g., GrubHub) and real-time taxi-calling service (e.g., Uber) in a small town/area with a small population, that pursue high reward/profit. These applications focus more on helping workers obtain high rewards. Besides, a start-up SC platform would sacrifice efficiency for task assignment effectiveness to attract more workers and reduce worker turnover.
2. BR+SA achieves good balance between efficiency and effectiveness (it is second only to OTA in terms of the total reward).
3. GTA is the most efficient algorithm, but it performs worse than other methods in effectiveness, which is applicable for the SC applications with a large number of tasks/workers. These applications need to improve the task assignment efficiency to ensure a considerable number of tasks to be assigned to suitable workers timely.
4. Considering the priority-aware Utility (PAU) of workers, GTA+PAU and BR+SA+PAU achieve more fair task assignments than their counterparts (i.e., GTA and BR+SA), respectively. GTA+PAU and BR+SA+PAU is suitable for the SC applications that pursue task assignment effectiveness and efficiency under fair. A typical example is cooperative SC applications where multiple workers perform a task collaboratively. It is necessary to distribute the rewards for these workers fairly.

7 Related work

Recent studies in Spatial Crowdsourcing (SC) make great efforts to solve different task assignment problems [4, 5, 8, 20, 26, 28, 29, 33, 35, 40, 45–47]. However, these studies adopt task assignment methods that assume centralized control and do not consider the coordination among workers, which incurs substantial computational costs especially in large-scale SC. The need for coordination increases greatly the system implementation difficulties and human efforts when attempting to apply existing methods in practice.

Additionally, the majority of SC studies focus on assigning tasks to single nearby workers based on different system optimization goals. However, in practice, complex tasks (such as monitoring traffic condition and cleaning rooms) occur in practice that require the involvement of multiple workers, called multiple task assignment. The present study goes further in this direction to address coalition-based task assignment by considering the rewarding and stability of worker coalitions. In our problem setting, workers are required to form coalitions to perform tasks through collaboration. Establishing worker coalitions is an important aspect of worker coordination and cooperation in SC, by which workers can enhance their combined capabilities of performing tasks and improve their utility.

Being an important factor for workers' satisfaction in crowdsourcing, fairness has been studied recently [9, 36, 39]. Basik et al. [9] offer a fair task allocation solution for crowdsourced delivery that focuses on distributive fairness, which is defined as the proximity between a worker's own input/output ratio and the input/output ratio of a referent, where the input to a worker is the total reward of the offers accepted and the output of a worker is the amount of reward earned. The notion of fairness in their study differs from our definition of fairness. Ye et al. [36] propose a fair task assignment framework that aims to maximize the minimum utility (i.e., the number of tasks assigned) for all workers, which is different from our goal. Moreover, it does not explore the priority awareness in the fair task assignment. Considering the degree of dissatisfaction for both workers (i.e., drivers) and tasks (i.e., passengers), Zhao et al. [39] propose a preference-aware task assignment problem for on-demand taxi dispatching, which is a single task assignment problem in which a task is assigned to only one worker. Their problem setting differs substantially from ours. Thus, their algorithm cannot solve our problem.

In the closest related research to ours, Cheng et al. [2] designs a game-theoretic approach to cooperative task assignment. However, their approach differs from ours in terms of the objectives and problem setting. First they [2] aim to maximize the total cooperation quality scores of assignments, while our goal is to maximize the overall rewards of workers with priority-aware fairness as a constraint. Further, they make the implicit assumption that workers are willing to voluntarily perform tasks assigned to them. In practice, workers are likely to be reluctant to perform assigned tasks without receiving payments or credits, as they face different participation costs (e.g., mobile device battery energy cost) [34], especially for complex tasks that require a group of workers to collaborate. In our problem, we take workers' rewards into consideration in order to motivate workers to perform tasks. Moreover, Cheng et al. [2] only obtain one Nash equilibrium among multiple Nash equilibriums through the best-response method, while we aim to achieve a better Nash equilibrium with higher total rewards through a combination of simulated annealing scheme and the best-response method.

8 Conclusion and future work

We study a novel problem, called Coalition-based Task Assignment (CTA), in spatial crowdsourcing, where an individual worker may not be able to accomplish a task independently because the task exceeds the capability of a single worker. Instead, workers are required to form stable coalitions with sufficient cumulative capabilities (or time) to perform the tasks. As the CTA problem is NP-hard, we propose different algorithms (including a greedy and an equilibrium-based algorithm) to efficiently and effectively assign tasks that maximize the overall rewards. The first algorithm assigns tasks to the nearby workers greedily and adopts an acceptance probability to achieve high-value task assignments. The equilibrium-based algorithm combines the best-response strategy and simulated annealing to find a Nash equilibrium that represents an approximately optimal task assignment. We further improve these two algorithms by integrating marginal contribution-based reward distribution and a priority-aware fairness mechanism to achieve fair task assignments. An extensive empirical study demonstrates that the proposed solutions are able to deliver near maximum total rewards with reasonably small running times and that the improved algorithms are able to find more fair assignments with lower average payoff differences. Future research directions include (i) to parallelize the solutions so that they scale to more data-intensive applications, (ii) to improve the efficiency of the equilibrium-based approaches by enabling lazy updating of the best responses, (iii) to generate an optimal task execution schedule for each worker to achieve the maximum reward, and (iv) to explore more efficient means of calculating marginal contribution-based values.

Acknowledgements This work is partially supported by NSFC (Nos. 61972069, 61836007, 61832017, 62272086), Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021), Municipal Government of Quzhou under Grant No. 2022D037, and Key Laboratory of Data Intelligence and Cognitive Computing, Longhua District, Shenzhen. It is also supported by Hong Kong Research Grants Council (No. 16202722), Natural Science Foundation of China (No. 62072125) and is partially conducted in the JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust.

Funding Open access funding provided by Aalborg University Library.

References

1. Chen, Z., Fu, R., Zhao, Z., Liu, Z., Xia, L., Chen, L., Cheng, P., Cao, C.C., Tong, Y., Zhang, C.J.: gmission: a general spatial crowdsourcing platform. PVLDB **7**(13), 1629–1632 (2014)
2. Cheng, P., Chen, L., Ye, J.: Cooperation-aware task assignment in spatial crowdsourcing. In: ICDE, pp. 1442–1453 (2019)
3. Cheng, P., Lian, X., Chen, L., Han, J., Zhao, J.: Task assignment on multi-skill oriented spatial crowdsourcing. TKDE **28**(8), 2201–2215 (2015)
4. Cheng, P., Lian, X., Chen, Z., Fu, R., Chen, L., Han, J., Zhao, J.: Reliable diversity-based spatial crowdsourcing by moving workers. PVLDB **8**(10), 1022–1033 (2015)
5. Cui, Y., Deng, L., Zhao, Y., Yao, B., Zheng, V.W., Zheng, K.: Hidden poi ranking with spatial crowdsourcing. In: SIGKDD, pp. 814–824 (2019)
6. D., F., J., T.: Game Theory. MIT Press (1991)
7. Deng, D., Shahabi, C., Zhu, L.: Task matching and scheduling for multiple workers in spatial crowdsourcing. In: SIGSPATIAL, pp. 2101–2110 (2015)
8. Deng, L., Lian, D., Huang, Z., Chen, E.: Graph convolutional adversarial networks for spatiotemporal anomaly detection. TNNLS **33**(6), 2416–2428 (2022)
9. Fuat, B., Bugra, G., Hakan, F., Kun-Lung, W.: Fair task allocation in crowdsourced delivery. CoRR **abs/1807.02987** (2018). http://arxiv.org/abs/1807.02987
10. Fudenberg, D., Levine, D.K.: The Theory of Learning in Games. MIT Press, New York (1998)
11. Gummidi, S.R.B., Pedersen, T.B., Xie, X.: Transit-based task assignment in spatial crowdsourcing. In: SSDBM, pp. 1301–1312 (2020)
12. Jong, S.D., Tuyls, K., Verbeeck, K.: Artificial agents learning human fairness. In: AAMAS, pp. 863–870 (2008)

13. Jong, S.D., Tuyls, K., Verbeeck, K., Roos, N.: Priority awareness: towards a computational model of human fairness for multi-agent systems. In: AAMAS, pp. 117–128 (2008)
14. Kaufman, D.E., Smith, R.L.: Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. JITS **1**(1), 1–11 (1993)
15. Kazemi, L., Shahabi, C.: Geocrowd: Enabling query answering with spatial crowdsourcing. In: SIGSPATIAL, pp. 189–198 (2012)
16. Kazemi, L., Shahabi, C., Chen, L.: Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In: SIGSPATIAL, pp. 314–323 (2013)
17. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)
18. Li, X., Zhao, Y., Guo, J., Zheng, K.: Group task assignment with social impact-based preference in spatial crowdsourcing. In: DASFAA, pp. 677–693 (2020)
19. Li, X., Zhao, Y., Zheng, K., Zhou, X.: Consensus-based group task assignment with social impact in spatial crowdsourcing. Data Sci. Eng. **5**(4), 375–390 (2020)
20. Li, Y., Zhao, Y., Zheng, K.: Preference-aware group task assignment in spatial crowdsourcing: a mutual information-based approach. In: ICDM, pp. 350–359 (2021)
21. Liu, Y., Dong, L., Marks, R.J.: Common control channel assignment in cognitive radio networks using potential game theory. In: WCNC, pp. 315–320 (2013)
22. Monderer, D., Shapley, L.S.: Potential games. Games Econ. Behav. **14**(1), 124–143 (1996)
23. Myerson, R.B.: Game Theory: Analysis of Conflict. Harvard University Press, New York (1997)
24. Nash, J.F., et al.: Equilibrium points in n-person games. Proc. Natl. Acad. Sci. **36**(1), 48–49 (1950)
25. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. MIT Press, New York (2007)
26. Song, T., Tong, Y., Wang, L., She, J., Yao, B., Chen, L., Xu, K.: Trichromatic online matching in real-time spatial crowdsourcing. In: ICDE, pp. 1009–1020 (2017)
27. Tong, Y., She, J., Ding, B., Wang, L.: Online mobile micro-task allocation in spatial crowdsourcing. In: ICDE, pp. 49–60 (2016)
28. Tong, Y., Wang, L., Zhou, Z., Chen, L., Du, B., Ye, J.: Dynamic pricing in spatial crowdsourcing: a matching-based approach. In: SIGMOD, pp. 773–788 (2018)
29. Tong, Y., Wang, L., Zhou, Z., Ding, B., Chen, L., Ye, J., Xu, K.: Flexible online task assignment in real-time spatial data. PVLDB **10**(11), 1334–1345 (2017)
30. Tong, Y., Zhou, Z., Zeng, Y., Chen, L., Shahabi, C.: Spatial crowdsourcing: a survey. PVLDB **29**(1), 217–250 (2020)
31. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2013)
32. Wang, Z., Li, Y., Zhao, K., Shi, W., Lin, L., Zhao, J.: Worker collaborative group estimation in spatial crowdsourcing. Neurocomputing **428**, 385–391 (2021)
33. Wang, Z., Zhao, Y., Chen, X., Zheng, K.: Task assignment with worker churn prediction in spatial crowdsourcing. In: CIKM (2021)
34. Xia, J., Zhao, Y., Liu, G., Xu, J., Zhang, M., Zheng, K.: Profit-driven task assignment in spatial crowdsourcing. In: IJCAI, pp. 1914–1920 (2019)
35. Ye, G., Zhao, Y., Chen, X., Zheng, K.: Task allocation with geographic partition in spatial crowdsourcing. In: CIKM (2021)
36. Ye, Q.C., Zhang, Y., Dekker, R.: Fair task allocation in transportation. OMEGA **68**, 1–16 (2017)
37. Zeng, Y., Tong, Y., Chen, L., Zhou, Z.: Latency-oriented task completion via spatial crowdsourcing. In: ICDE, pp. 317–328 (2018)
38. Zhao, B., Xu, P., Shi, Y., Tong, Y., Zhou, Z., Zeng, Y.: Preference-aware task assignment in on-demand taxi dispatching: An online stable matching approach. In: AAAI, pp. 2245–2252 (2019)
39. Zhao, B., Xu, P., Shi, Y., Tong, Y., Zhou, Z., Zeng, Y.: Preference-aware task assignment in on-demand taxi dispatching: an online stable matching approach. In: AAAI, vol. 33, pp. 2245–2252 (2019)
40. Zhao, Y., Chen, X., Deng, L., Kieu, T., Guo, C., Yang, B., Zheng, K., Jensen, C.S.: Outlier detection for streaming task assignment in crowdsourcing. In: WWW (2022)
41. Zhao, Y., Guo, J., Chen, X., Hao, J., Zhou, X., Zheng, K.: Coalition-based task assignment in spatial crowdsourcing. In: ICDE (2021)
42. Zhao, Y., Li, Y., Wang, Y., Su, H., Zheng, K.: Destination-aware task assignment in spatial crowdsourcing. In: CIKM, pp. 297–306 (2017)
43. Zhao, Y., Xia, J., Liu, G., Su, H., Lian, D., Shang, S., Zheng, K.: Preference-aware task assignment in spatial crowdsourcing. In: AAAI, pp. 2629–2636 (2019)
44. Zhao, Y., Zheng, K., Cui, Y., Su, H., Zhu, F., Zhou, X.: Predictive task assignment in spatial crowdsourcing: a data-driven approach. In: ICDE, pp. 13–24 (2020)
45. Zhao, Y., Zheng, K., Guo, J., Yang, B., Pedersen, T.B., Jensen, C.S.: Fairness-aware task assignment in spatial crowdsourcing: game-theoretic approaches. In: ICDE, pp. 265–276 (2021)
46. Zhao, Y., Zheng, K., Li, Y., Su, H., Liu, J., Zhou, X.: Destination-aware task assignment in spatial crowdsourcing: a worker decomposition approach. In: TKDE, pp. 2336–2350 (2019)
47. Zhao, Y., Zheng, K., Yin, H., Liu, G., Fang, J., Zhou, X.: Preference-aware task assignment in spatial crowdsourcing: from individuals to groups. In: TKDE (2020)