Profit Optimization in Spatial Crowdsourcing: Effectiveness and Efficiency

Yan Zhao, Kai Zheng, *Senior Member, IEEE*, Yunchuan Li, Jinfu Xia, Bin Yang, Torben Bach Pedersen, *Senior Member, IEEE*, Rui Mao, Christian S. Jensen, *Fellow, IEEE*, and Xiaofang Zhou, *Fellow, IEEE*

Abstract—In Spatial crowdsourcing, mobile users perform spatio-temporal tasks that involve travel to specified locations. Spatial crowdsourcing (SC) is enabled by SC platforms that support mobile worker recruitment and retention, as well as task assignment, which is essential to maximize profits that are accrued from serving task requests. Specifically, how to best achieve task assignment in a cost-effective manner while contending with spatio-temporal constraints is a key challenge in SC. To address this challenge, we formalize and study a novel Profit-driven Task Assignment problem. We first establish a task reward pricing model that takes into account the temporal constraints (i.e., expected completion time and deadline) of tasks. Then we adopt an optimal algorithm based on tree decomposition to achieve an optimal task assignment and propose greedy algorithms based on Random Tuning Optimization to improve the computational efficiency. To balance effectiveness and efficiency, we also provide a heuristic task assignment algorithm based on Ant Colony Optimization that assigns tasks by simulating behavior of ant colonies foraging for food. Finally, we conduct extensive experiments using real and synthetic data, offering detailed insight into effectiveness and efficiency of the proposed methods.

Index Terms—Spatial crowdsourcing, task assignment, profit

1 Introduction

Recent years have witnessed a revolution in Spatial Crowdsourcing (SC), where people with mobile connectivity can accept and accomplish a variety of location-based tasks [9], [21], [27], [28], [29], [30], [31], [33], [42]. A typical SC system encompasses a platform, task requesters, and mobile workers. The platform provides task assignment services to task requesters, and enables workers to serve requests. Most existing research in SC targets task assignment while adopting different optimization strategies [3], [5], [43]. Although task assignment has been studied extensively, problems such as how to optimize the profits achieved by an SC platform remain unsolved.

A primary quality of an SC platform is its inherent cost effectiveness. In other words, it is essential for an SC platform to be able to perform task assignments that maximize profits. Although several SC applications have been proposed, these generally assume that an SC platform assigns tasks to mobile workers on a voluntary basis, without considering the profit of the SC system [6], [14]. However, effective non-monetary incentive schemes for tedious and repetitive work are often difficult to engineer [26], and such schemes are often unrealistic for commercial SC platforms due to their profit-driven nature. Moreover, workers are unwilling to perform assigned tasks without pay as performing tasks incurs costs (e.g., mobile device battery energy cost for sensing and data processing). Several profit-based task assignment mechanisms have been developed for crowdsourcing. For instance, considering the profit of a platform, Yang et al. [38] provide incentive mechanisms for mobile crowdsourcing based on platform-centric and user-centric system models, but they focus mainly on improving computational efficiency rather than improving the profit. A recent study [25] proposes a profit maximizing truthful auction mechanism, where the platform acts as an auctioneer and workers act as sellers that submit bids to the platform. However, they focus on improving the total profit of a platform by mean of incentive mechanisms instead of paying attention to the task assignment process. Sarker et al. [24] develop a workload allocation policy that enables reasonable trade-offs between worker utility and platform profit. Nevertheless, this study focuses on sensing tasks and divides a task into subtasks of uniform size, and workers perform the subtasks independently without cooperation.

- *Yan Zhao, Bin Yang, Torben Bach Pedersen, and Christian S. Jensen are with the Department of Computer Science, Aalborg University, 9220 Aalborg, Denmark. E-mail: {yanz, byang, tbp, csj}@cs.aau.dk.*
- *Kai Zheng and Yunchuan Li are with the School of Computer Science and Engineering & Shenzhen Institute for Advanced Study, University of Electronic Science and Technology Chinat, Chengdu, Sichuan 610056, China. E-mail: zhengkai@uestc.edu.cn, liyunchuan@std.uestc.edu.cn.*
- *Jinfu Xia is with Soochow University, Suzhou, Jiangsu 215000, China. E-mail: viing937@gmail.com.*
- *Rui Mao is with Shenzhen University, Shenzhen, Guangdong Province 518061, China. E-mail: mao@szu.edu.cn.*
- *Xiaofang Zhou is with the Hong Kong University of Science and Technology, Hong Kong. E-mail: zxf@cse.ust.hk.*

Manuscript received 31 December 2020; revised 22 June 2022; accepted 3 November 2022. Date of publication 8 November 2022; date of current version 21 June 2023.
This work was supported in part by NSFC under Grants 61972069, 61836007 and 61832017, in part by Shenzhen Municipal Science and Technology R&D Funding Basic Research Program under Grant JCYJ20210324133607021, and in part by the JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust.
(Corresponding author: Kai Zheng.)
Recommended for acceptance by W. Ku.
Digital Object Identifier no. 10.1109/TKDE.2022.3220360

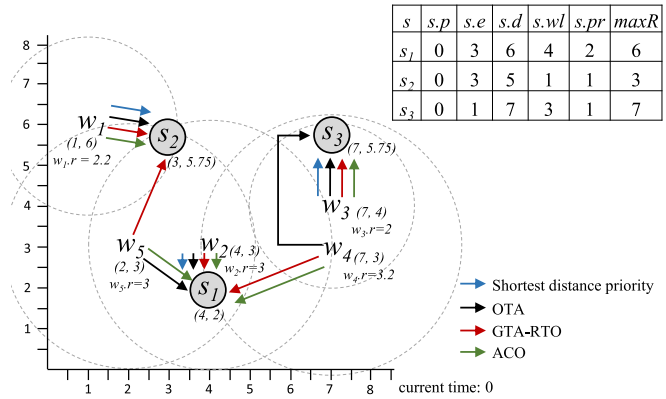


Fig. 1. Running example.

Li et al. [18] propose an online optimization framework to maximize the profit in mobile crowdsourcing. In doing so, they transform integer programming into fraction programming and construct a pseudo-tree for use in task assignment. But they disregard spatial constraints that play an essential role in SC when assigning tasks.

We investigate profit-based task assignment in SC, where each task has a certain workload that can be assigned to one or more workers to move to the location of the task physically and complete the task, e.g., maintenance tasks, leaflet distribution, and holding a barbecue party, cooperatively. For example, the workload of a leaflet distribution task is shared among workers by assigning each worker a certain number of leaflets to distribute when a worker arrives at the task's location. If all leaflets are distributed on time, the workers and the SC platform obtain the task reward and profit. Fig. 1 exemplifies profit-based task assignment, considering five workers ($\{w_1, ..., w_5\}$) and three tasks ($\{s_1, s_2, s_3\}$). Each worker has a current location and a reachable distance range ($w.r$). For example, worker w_1 is located at (1,6) and has reachable distance 2.2. Each task is published and expires at particular times ($s.p$ and $s.d$, respectively) and is labeled with its expected completion time ($s.e$), workload ($s.wl$), and reward information (i.e., penalty rate $s.pr$ and maximum reward $s.maxR$). The problem is to assign tasks to suitable workers so as to maximize the total platform profit. In SC, it is intuitive to assign tasks to nearby workers without violating spatio-temporal constraints (i.e., workers' reachable distance and tasks' expiration time), which is referred to as the *shortest distance priority* algorithm. With this algorithm, we obtain the task assignment $\{(s_1, w_2),(s_2, w_1), (s_3, w_3)\}$ (shown with blue arrows in Fig. 1) that yields a platform profit of 6.58. The calculation of task reward is based on the reward pricing model to be presented in Section 3, and we assume that the platform gets 80% of task rewards. However, this algorithm leaves w_4 and w_5 unassigned, which indicates that the profits are sub-optimal.

We propose a profit-driven task assignment framework that links task assignment with its economic performance. We first formulate the Profit-driven Task Assignment (PTA) problem and show that it is NP-hard. We then establish a reward pricing model that takes into account the temporal constraints of tasks and where the total reward is tied to the payments by task requesters and the processing time of tasks. When it comes to task assignment, we introduce an exact tree-decomposition-based algorithm, called Optimal Task Assignment (OTA), that finds assignments that yield the optimal total platform profit. To achieve computational efficiency, we propose a Greedy Task Assignment (GTA) algorithm that gives priority to tasks with the higher reward per unit of work and assigns tasks to the closest workers, ensuring that tasks can be finished before their expected completion times. Further, GTA with coarse and fine-grained Random Tuning Optimization (GTA-RTO) strategies are developed to prune non-promising worker-task assignment pairs. The black and red arrows in Fig. 1 depict the task assignments by the OTA and GTA-RTO algorithms that generate profits of 9.60 and 8.07.

The conference version of this work [37] provides OTA and GTA-RTO algorithms that address the PTA problem. However, the OTA algorithm is time consuming, and the GTA-RTO algorithm achieves relatively low profits for SC platforms, especially for commercial SC platforms. An SC platform tends to prioritize high profits even if some efficiency must be sacrificed. Therefore, to support the diverse effectiveness and efficiency requirements of SC platforms, we provide a tailor-made task assignment method based on Ant Colony Optimization (ACO), which can achieve a better balance between effectiveness and efficiency. An ACO algorithm, first proposed by Dorigo et al. [12], takes inspiration from the foraging behavior of some ant species. The ants deposit pheromones on the ground to indicate favorable paths that should be followed by other members of the colony. Due to their advantages, ACO algorithms are applied widely for solving combinatorial optimization problems, such as the traveling salesman problem and assignment problem [10], [11], [23], [48]. To solve our problem using ACO, we utilize the foraging behavior of a set of artificial ants to simulate the task assignment process. Ants aim to find short paths between tasks and workers in task-worker assignment pairs according to a stochastic mechanism that is biased by pheromones (left by previous ants) and heuristic information (determined by travel distance). A resulting task assignment consists of a set of short paths (i.e., task-worker pairs). When applying ACO to the example in Fig. 1, we obtain the task assignment $\{(s_1, \{w_2, w_4, w_5\}), (s_2, \{w_1\}), (s_3, \{w_3\})\}$ with a profit of 9.04, which exceeds the profit obtained by GTA-RTO.

The major value-added extensions over our preliminary work [37] are three-fold.

1) We identify and study in depth a limitation in our previous algorithms, thus enabling improved trade-offs between effectiveness and efficiency in task assignment.
2) We provide an ACO-based task assignment algorithm that solves the PTA problem, achieving a better balance between effectiveness (second to OTA) and efficiency (second to GTA-RTO).
3) We report on extensive experiments that offer detailed insight into the effectiveness and efficiency of the paper's proposals. In particular, the CPU time of ACO is only 0.02%–15.89% of that of OTA, while being able to achieve up to 97.60% of the profit of OTA. ACO can improve the overall profit ratio by

TABLE 1
Summary of Notation

Notation	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
S	Spatial task set
w	Worker
$w.l$	Location of worker w
$w.r$	Reachable distance of worker w
W	Worker set
$AWS(s)$	Available worker set for task s
$t(a, b)$	Travel time from location a to location b
$d(a, b)$	Travel distance from location a to location b
$P_{AWS(s)}$	The profit obtained by an SC platform after workers in $AWS(s)$ completing task s
$R_{AWS(s)}$	The reward obtained by workers in $AWS(s)$ by completing task s
$OptAWS(s)$	Optimal available worker set for task s
A	A spatial task assignment
$A.P$	Total profit of spatial task assignment A
$\mathbb{A}$	Spatial task assignment set

up to 6.50% compared with GTA-RTO when they consume the same CPU time under appropriate settings of the numbers of algorithm iterations and ants.

The remainder of the paper is organized as follows. Section 2 introduces preliminary concepts and the PTA problem. We then propose the reward pricing model in Section 3. The task assignment algorithms (including the optimal, greedy, and ACO-based algorithms) are presented in Section 4. We report on the empirical study in Section 5, followed by a coverage of related work in Section 6. Section 7 concludes the paper.

2 Preliminaries and Problem Statement

We first introduce preliminaries in the context of reward-based task assignment, where tasks are assigned to workers who will receive a certain reward for completing a task, in SC adopting the Server Assigned Tasks (SAT) mode [15]. Then we formulate the Profit-driven Task Assignment (PTA) problem and prove that it is NP-hard. Table 1 summarizes notation used throughout the paper.

Definition 1 (Spatial Task). *A spatial task, denoted by $s = (l, p, e, d, wl, maxR, pr)$, 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 required to finish a task to denote $s.wl$). 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 reward.*

Definition 2 (Worker). *A worker, denoted by $w = (l, r)$, encompasses a location $w.l$ and a reachable distance $w.r$. The reachable range of worker w is a circle with center $w.l$ and radius $w.r$, within which w can accept assignments.*

Definition 3 (Available Worker Set). *Given a task s to be assigned and a set of workers in the vicinity of s, an available worker set for task s, denoted as $AWS(s)$, should satisfy three conditions: $\forall w \in AWS(s)$*

1) *worker w is able to arrive at the location of task s before its deadline, i.e., $t_{now} + t(w.l, s.l) < s.d$,*
2) *task s is located in the reachable range of worker w, i.e., $d(w.l, s.l) \leq w.r$, and*
3) *all workers in $AWS(s)$ have enough time to complete task s together before it expires, i.e., $\sum_{w \in AWS(s)}(s.d - (t_{now} + t(w.l, s.l))) \geq s.wl$,*

where t_{now} is the current time, $t(a, b)$ 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 three conditions guarantee that workers in an available worker set can travel from their origins to the location of their reachable task s before it expires and can complete the workload of s together.

Definition 4 (Platform Profit). *Given a task s to be assigned and an available worker set $AWS(s)$, the profit of the SC platform can be computed as $P_{AWS(s)} = \alpha R_{AWS(s)}$, where $P_{AWS(s)}$ and $R_{AWS(s)}$ $(0 \leq R_{AWS(s)} \leq s.maxR)$ are the profit of the platform to finish s and the reward for s (i.e., the payment that the task requester provides), respectively, when assigning task s to workers in $AWS(s)$. Parameter α $(0 < \alpha < 1)$ indicates the fraction of the task reward that the platform obtains. The calculation of the reward for s, $R_{AWS(s)}$, is elaborated in Section 3.*

Note that we simply assume that the profit of an SC platform is proportional to the reward (e.g., the profit is 80% of the reward when $\alpha = 80\%$) and that the remaining reward is allocated to workers. Real SC platforms, e.g., real-time ride-hailing services (e.g., Uber[1]) and on-wheel meal-ordering services (e.g., GrubHub[2]), make money by taking a proportion of the commission from task requesters (who publish tasks) for each task assignment. SC platforms provide task assignment service to workers and task requesters. This way, platforms have a strong impact on the organisation of work and, above all, on the relationships between workers and task requesters. In a nutshell, the platforms allow individuals, families, or companies in need of a service to hire a worker who is willing to offer the service, under the guise of enhanced flexibility and at the task requester's premises. Therefore, it is reasonable for the platforms to be rewarded proportionally to the workers' rewards in order to incentivize them. Our solutions focus on the profit of the SC platforms and do not consider the profit/reward allocation mechanism of workers. However, the algorithms we propose are independent of this assumption and can handle different profit/reward allocation mechanisms.

Definition 5 (Optimal Available Worker Set (OptAWS)). *An available worker set, $AWS(s)$, is optimal if every proper subset of it can only achieve a less-than-$P_{AWS(s)}$ platform profit.*

1. https://www.uber.com/
2. https://get.grubhub.com/

Note that more than one optimal AWS may exist for a given task s. For example, in Fig. 1, $\{w_1\}$, $\{w_1, w_2\}$, and $\{w_1, w_5\}$ are optimal for task s_2, while $\{w_1, w_2, w_5\}$ is not optimal for task s_2 since one of its proper subsets, $\{w_1, w_2\}$, can achieve the same platform profit.

Definition 6 (Spatial Task Assignment). *Given a set of workers W and a set of tasks S, a spatial task assignment A consists of a set of pairs of a task and an OptAWS for the task: $(s_1, OptAWS(s_1))$, $(s_2, OptAWS(s_2))$,..., $(s_{\mathcal{N}}, OptAWS(s_{\mathcal{N}}))$, where $\bigcap_{i=1}^{i=\mathcal{N}} OptAWS(s_i) = \emptyset$, and $\mathcal{N}$ denotes the number of tasks.*

Let $A.P$ denote the total profit of the SC platform for task assignment A, i.e., $A.P = \sum_{(s,OptAWS(s))\in A} P_{OptAWS(s)}$, and let $\mathbb{A}$ denote all possible assignments. The problem investigated can be stated as follows.

Definition 7 (PTA Problem). *Given a worker set W and a task set S, the PTA problem is to find the globally optimal assignment A_{opt}, such that $\forall A_i \in \mathbb{A}$, $A_i.P \leq A_{opt}.P$.*

Next, we prove the hardness of the PTA problem.

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

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

A given 0-1 knapsack problem instance can be transformed to an instance of the PTA problem as follows: we are given a task set S of n tasks, in which each task s_i is associated with the publication time $s_i.p = 0$, expected completion time $s_i.e = 1$, deadline $s_i.d = 1$, workload $s_i.wl = m_i$, and maximum reward $s_i.maxR = v_i$. We are also given a set W of M workers. Moreover, all tasks and workers are situated at the same location. Thus, in this instance of the assignment problem, it is only possible to assign m_i workers to task s_i in order to get reward v_i.

If we can solve the PTA problem efficiently (i.e., in polynominal time), we can solve a 0-1 knapsack problem instance by transforming it to the corresponding PTA problem instance and then solve that instance efficiently. This contradicts the fact that the 0-1 knapsack problem is NP-hard [35], and so there cannot be an efficient solution to the PTA problem that is then NP-hard. □

3 Reward Pricing Model Construction

In order to develop a cost-effective task assignment algorithm, a reasonable reward pricing model has to be established that takes into account the temporal constraints of tasks. We thus design a Reward Pricing Model (RPM) based on the intuition that taking longer to complete a task (including waiting time and task duration, covering the time from when a task is published to when it is finished) increases the probability of task failure, which in turn reduces the rewards and profit for the SC platform. In particular, we consider a single task s with a publication time $s.p$, an expected completion time $s.e$, a deadline $s.d$, a required workload $s.wl$, a maximum reward $s.maxR$, and a penalty rate $s.pr$. RPM focuses on the task completion time and actual reward (i.e., the requester's actual payment in return for completing the task), which is illustrated in Fig. 2.

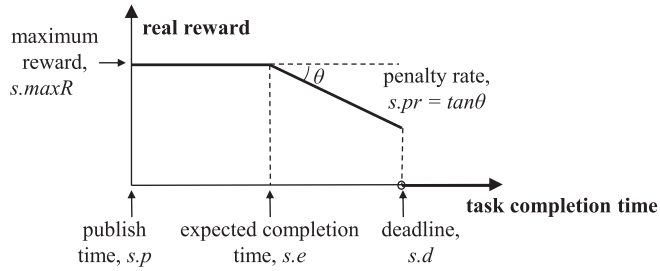


Fig. 2. Task reward pricing model.

Using the above properties of a task, the RPM expresses the actual payment that will be made in return for completing task s, as shown in Equation (1).

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

where $R_{AWS(s)}$ represents the actual reward of task s, and $s.t_e$ denotes the completion time of s given the available worker set AWS.

In order to calculate $s.t_e$, we let $s.t_s$ denote the start time (i.e., time of assignment) of s, let $T(AWS(s))$ denote the duration of task s (i.e., the time from assignment to completion), and let $w.wl(AWS(s)) > 0$ denote the workload contribution (a time duration) of w when task s is performed by workers in $AWS(s)$. Fig. 3 illustrates the workload allocation of an available worker set $\{w_2, w_5\}$ for task s_1 based on the example in Fig. 1. It follows that, for each worker w in $AWS(s)$, the task duration is equal to w's travel time plus w's workload contribution, i.e.,

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

Summing up the right side over all workers in $AWS(s)$, we have

$$\begin{aligned} &T(AWS(s)) \\ &= \frac{\sum_{w\in AWS(s)} t(w.l, s.l) + \sum_{w\in AWS(s)} w.wl(AWS(s))}{|AWS(s)|} \end{aligned} \tag{3}$$

Given that $s.wl = \sum_{w\in AWS(s)} w.wl(AWS(s))$, we get

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

Finally, $s.t_e$ can be calculated as $s.t_e = s.t_s + T(AWS(s))$, and each worker's workload can be calculated as $w.wl(AWS(s)) = T(AWS(s)) - t(w.l, s.l)$, where $T(AWS(s))$ denotes the duration of task s. According to the RPM in Fig. 2, we make two observations.

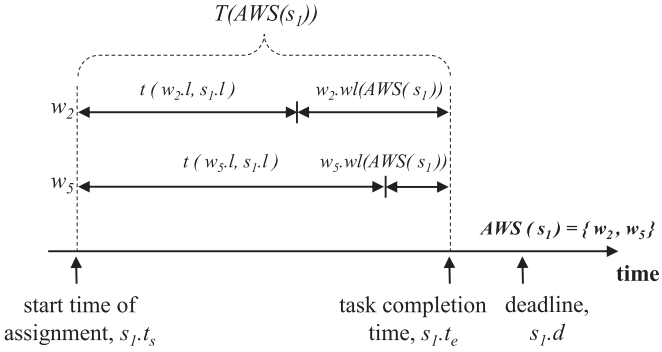


Fig. 3. Workload allocation of available worker set $\{w_2, w_5\}$ for task s_1.

1) Since we require $w.wl(AWS(s)) > 0$, if the travel time of a worker w ($w \in W$) exceeds the task duration, i.e., $t(w.l, s.l) \geq T(AWS(s))$, then worker w has no contribution to task s and should be removed from $AWS(s)$.
2) When the workers in an available worker set $AWS(s)$ can cooperate to finish a task s before its expected completion time $s.e$, meaning that they obtain the maximum reward ($s.maxR$) for completing s, adding more workers to $AWS(s)$ cannot yield a higher reward.

In other words, adding workers to $AWS(s)$ does not necessarily mean an earlier completion time or a higher total reward, which explains why we propose the concept of an optimal available worker set in Definition 5.

4 Task Assignment

4.1 Optimal Task Assignment (OTA)

The globally optimal task assignment assigns a possible available worker set or a null worker set to each task in a manner that maximizes the profit for the SC platform. In this section, we apply a tree-decomposition-based algorithm [42], [46] to achieve an optimal task assignment. We do this in three steps:

1) Find all reachable workers for each task. The reachable worker set for a task s, denoted as $RW(s)$, must satisfy the following conditions: $\forall w \in RW(s)$, $t_{now} + t(w.l, s.l) < s.d \wedge d(w.l, s.l) \leq w.r$. These two conditions guarantee that w can reach s before it expires and s is in the reachable range of w.
2) Find $OptAWS$s for each task. Given the reachable worker set for each task, we utilize a dynamic programming algorithm [8] that iteratively expands the sets of tasks in ascending order of the set size to find all $OptAWS$s.
3) Apply the tree-decomposition-based algorithm [42], [46] to find an optimal task assignment. In particular, we first utilize a graph to represent task dependencies, where two tasks sharing reachable workers are dependent and are connected by an edge in the graph. Then we use tree decomposition to separate all tasks into clusters, each of which is a maximal clique of the graph, and we organize them into a tree structure such that tasks in sibling nodes do not share the same reachable workers. Then the tree can be traversed in a depth-first manner to find an optimal assignment. The process is omitted due to space limit.

When applying the OTA algorithm to the example in Fig. 1, we get a profit of 9.60 for the SC platform with the task assignment $\{(s_1, \{w_2, w_5\}), (s_2, \{w_1\}), (s_3, \{w_3, w_4\})\}$.

4.2 Greedy Task Assignment With Random Tuning Optimization (GTA-RTO)

4.2.1 Basic Greedy Task Assignment

For the sake of efficiency, we propose a basic Greedy Task Assignment (GTA) solution to the PTA problem that gives higher priority to tasks with higher reward per unit of work and to workers with low travel cost.

GTA, shown in Algorithm 1, takes a worker set W and a task set S as input, and it returns a task assignment set A, an unassigned worker set W', and an unassigned task set S'. After initialization (line 1), we sort the tasks in S' descendingly according to their reward per unit of work, i.e., $\frac{s.maxR}{s.wl}$ (line 2). Then for each task s in S', we try to assign the first arriving worker (i.e., the closest worker) in W' to s until s can be finished before its expected completion time $s.e$ (lines 3–13). Specifically, if there are insufficient workers to complete task s, s is skipped (lines 7–8); otherwise, the closest worker in W' is assigned to task s (lines 9–10). When the workload of s is finished by the assigned workers, the task assignment A is updated (lines 11–12). The time complexity of GTA is $O(max\{\mathcal{N} \cdot \log \mathcal{N}, \mathcal{N} \cdot \mathbb{N} \cdot \log \mathbb{N}\})$, where $\mathcal{N}$ denotes the number of tasks and $\mathbb{N}$ denotes the number of workers. When applied to the example in Fig. 1, GTA obtains the task assignment $\{(s_2, \{w_1, w_2\}), (s_3, \{w_3, w_4\})\}$ that yields a profit of 5.8.

Algorithm 1. GTA

Input: W, S
output: A, W', S'
1: $A \leftarrow \emptyset, W' \leftarrow W, S' \leftarrow S$
2: Sort tasks in S' according to $\frac{s.maxR}{s.wl}$ descendingly
3: **for** each $s \in S'$ **do**
4: $AWS(s) \leftarrow \emptyset$
5: Sort workers in W' according to their arrival time ascendingly
6: **while** s is not finished before its expected completion time **do**
7: **if** W' is $\emptyset$ **then**
8: break
9: $w \leftarrow$ the closest worker in W'
10: $AWS(s) \leftarrow AWS(s) \cup \{w\}$
11: **if** s is finished **then**
12: $A \leftarrow A \cup \{(s, AWS(s))\}$
13: $S' \leftarrow S' - \{s\}; W' \leftarrow W' - AWS(s)$
14: **Return** A, W', S'

4.2.2 Random Tuning Optimization

Although GTA is a polynomial-time greedy algorithm that finds a good task assignment set A, some weaknesses exist in GTA. First, GTA always tries to complete a task before its expected completion time while ignoring the penalty rate and deadline of the task, which also impact the final profit. Moreover, GTA assigns the closest workers without considering the time utilization ratios of workers. For instance, when a task is distant from all workers, assigning the closest

workers to perform this task may yield an overall low profit for the platform.

To address these limitations, we improve GTA with Random Tuning Optimization (obtaining GTA-RTO), which contains two tuning strategies: *coarse tuning* and *fine tuning*. Coarse tuning randomly abandons low-value tasks, i.e., tasks with low time utilization ratio (measured by travel time ratio and rate of return) of workers, and fine tuning reassigns partial workers randomly to find a better task assignment.

GTA-RTO is detailed in Algorithm 2, which starts from an assignment achieved by GTA (line 1). Then the current task assignment A', unassigned worker set W', and unassigned task set S' are modified in a tuning loop (lines 4–14), where the Coarse Tuning (CT) and Fine Tuning (FT) algorithms are invoked in order, followed by GTA. Specifically, we first invoke CT with A' as input to generate an updated A', unassigned worker set W'_C, and unassigned task set S'_C (line 6). Then global unassigned worker set W' and task set S' are updated (line 7). Similarly, we update A', W', and S' by invoking FT (lines 8–9). Taking the global unassigned worker set W' and task set S' as input, GTA is invoked to generate a task assignment A_G (line 10). Subsequently, a new task assignment is achieved as the union of A' and A_G (line 11). If the profit of A' exceeds that of A (i.e., the current task assignment), we replace A with A' (lines 12–13). Algorithm 2 stops when there is no improvement, i.e., $A.P^k = A.P^{k-1}$, where $A.P^k$ denotes the profit of task assignment A in the kth iteration.

Algorithm 2. GTA-RTO

Input: W, S
output: A
1: $A', W', S' \leftarrow GTA(W, S)$
2: $A \leftarrow A'$
3: $k \leftarrow 0$
4: **repeat**
5: $k = k + 1$;
6: $A', W'_C, S'_C \leftarrow CT(A')$
7: $W' \leftarrow W' \cup W'_C$; $S' \leftarrow S' \cup S'_C$
8: $A', W'_F, S'_F \leftarrow FT(A')$
9: $W' \leftarrow W' \cup W'_F$; $S' \leftarrow S' \cup S'_F$
10: $A_G, W', S' \leftarrow GTA(W', S')$
11: $A' \leftarrow A' \cup A_G$
12: **if** $A'.P > A.P$ **then**
13: $A \leftarrow A'$
14: **until** $A.P^k = A.P^{k-1}$
15: /*$A.P^k$ denotes the profit of task assignment A in the kth iteration*/
16: **Return** A

Coarse Tuning (CT). CT aims to disregard low-value tasks, where the value of a task is measured by the travel time ratio and rate of return. More specifically, the probability $P^{CT}_{AWS(s)}(s)$ that task s (assigned to workers in available worker set $AWS(s)$) will be abandoned for its low value in CT is defined as follows:

$$P^{CT}_{AWS(s)}(s) = p^{CT}_m + p^{CT}_t \cdot \frac{\sum_{w \in AWS(s)} t(w.l, s.l)}{T(AWS(s)) \cdot |AWS(s)|} + p^{CT}_r \cdot \left(1 - \frac{R_{AWS(s)}}{s.maxR}\right), \tag{5}$$

where $p^{CT}_m + p^{CT}_t + p^{CT}_r = 1$, $t(w.l, s.l)$ denotes the travel time between $w.l$ and $s.l$, $T(AWS(s))$ is the task duration of task s with available worker set $AWS(s)$, and $R_{AWS(s)}$ is the reward of task s when it is finished by workers in $AWS(s)$. Next, $\frac{\sum_{w \in AWS(s)} t(w.l,s.l)}{T(AWS(s)) \cdot |AWS(s)|}$ represents the travel time ratio of workers in $AWS(s)$ to complete task s, and $\frac{R_{AWS(s)}}{s.maxR}$ is the rate of return. Further, p^{CT}_m is the minimum probability that a task is abandoned. Finally, p^{CT}_t and p^{CT}_r are parameters that control the contributions of the time utilization ratio of workers and the rate of return.

Algorithm 3. CT

Input: A
output: A_C, W_C, S_C
1: $A_C \leftarrow A$; $W_C \leftarrow \emptyset$; $S_C \leftarrow \emptyset$
2: **for** each task-AWS pair $(s, AWS(s)) \in A_C$ **do**
3: Calculate $P^{CT}_{AWS(s)}(s)$ based on Equation (5)
4: **if** $P^{CT}_{AWS(s)}(s) > \zeta$ **do**
5: $A_C \leftarrow A_C - (s, AWS(s))$
6: $S_C \leftarrow S_C \cup s$; $W_C \leftarrow W_C \cup AWS(s)$
7: **Return** A_C, W_C, S_C

CT is detailed in Algorithm 3, which takes a task assignment A as input and generates an updated task assignment A_C that abandons low-value tasks. In particular, we first calculate the probabilities of abandoning each task s in A_C (lines 2–3) and remove the task-AWS pair $(s, AWS(s))$ from A_C if the probability exceeds ζ, which is chosen randomly from the range [0,1] (lines 4–5).

Fine Tuning (FT). In the FT processing, no task is completely abandoned, and only few workers are reassigned, ensuring that tasks can be completed before their deadline. The probability $P^{FT}_{AWS(s)}(s, w)$ that worker w ($w \in AWS(s)$) assigned to task s is reassigned is defined as follows:

$$P^{FT}_{AWS(s)}(s, w) = p^{FT}_m + p^{FT}_t \cdot \frac{t(w.l, s.l)}{T(AWS(s))} \tag{6}$$

$$w \in AWS(s),$$

where $\frac{t(w.l,s.l)}{T(AWS(s))}$ is the ratio between the travel time of w and the task duration of s, called travel time ratio of w. Further, p^{FT}_m is the minimum probability that a worker is reassigned, p^{FT}_t captures how much the travel time ratio of worker w affects the probability, and $p^{FT}_m + p^{FT}_t = 1$.

FT is detailed in Algorithm 4 that takes a task assignment as input and outputs an updated task assignment A_F, an unassigned worker set W_F, and an unassigned task set S_F. First, we calculate the probability that each task s in A_F is abandoned based on Equation (5) (line 4) and the reassignment probability of each assigned worker $w \in AWS(s)$ based on Equation (6) (line 5). Subsequently, we randomly remove worker w from the current task assignment according to the reassignment probability $P^{FT}_{AWS(s)}(s, w)$ (i.e., a worker with a higher probability is more likely to be removed) when the abandoned probability exceeds a random number ζ (from range [0,1]) and s can be finished without w (lines 6–10). The task assignment A_F is updated accordingly (line 12) and is output (line 13).

The time complexity of both Algorithms 3 and 4 is $O(N_{(s,AWS(s))})$, where $N_{(s,AWS(s))}$ denotes the number of $task$-AWS pairs. Therefore, the time complexity of Algorithm 2 is $O(K \cdot \max\{\mathcal{N} \cdot \log \mathcal{N}, \mathcal{N} \cdot \mathbb{N} \cdot \log \mathbb{N}, N_{(s,AWS(s))}\})$, where K denotes the number of iterations, $\mathcal{N}$ denotes the number of tasks, and $\mathbb{N}$ denotes the number of workers. In the example in Fig. 1, the GTA-RTO algorithm returns the task assignment $\{(s_1, \{w_2, w_4\}), (s_2, \{w_1, w_5\}), (s_3, \{w_3\})\}$ with a profit of 8.07, which is 84.06% of that obtained by OTA.

Algorithm 4. FT

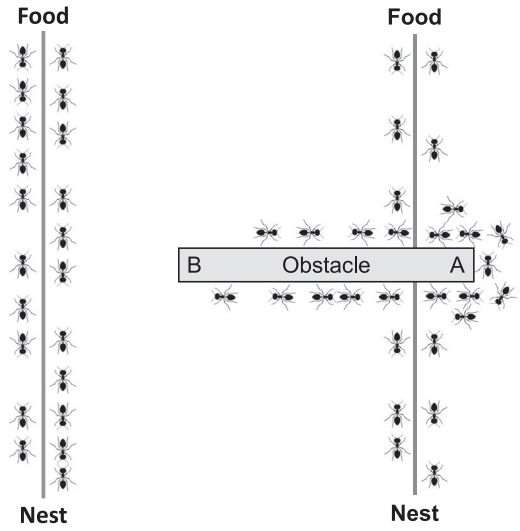
Input: A
output: A_F, W_F, S_F
1: $A_F \leftarrow A$; $W_F \leftarrow \emptyset$; $S_F \leftarrow \emptyset$
2: **for** each task-AWS pair $(s, AWS(s)) \in A_F$ **do**
3: $A_F \leftarrow A_F - (s, AWS(s))$
4: Calculate $p^{CT}_{AWS(s)}(s)$ based on Equation (5)
5: Calculate $P^{FT}_{AWS(s)}(s, w)$ for each worker $w \in AWS(s)$ based on Equation (6)
6: **while** $p^{CT}_{AWS(s)}(s) > \zeta$ **do**
7: Randomly choose a worker w from $AWS(s)$ according to $P^{FT}_{AWS(s)}(s, w)$
8: **if** s cannot be completed without w **then**
9: break
10: $AWS(s) \leftarrow AWS(s) - \{w\}$
11: $W_F \leftarrow W_F \cup \{w\}$
12: $A_F \leftarrow A_F \cup (s, AWS(s))$
13: **Return** A_F, W_F, S_F

4.3 Ant Colony Optimization (ACO) Based Task Assignment

Ant Colony Optimization (ACO) is a heuristic evolutionary algorithm that offers an alternative means of solving complicated combinatorial optimization problems [12]. ACO can achieve excellent performance via self-adaption and may thus yield good performance at solving optimization problems. Since the PTA problem is a combinatorial optimization problem, which aims to find an optimal task assignment from a finite set of task assignments, we adopt ACO to solving it.

4.3.1 Inspiration of ACO: Ant Foraging

ACO is based on the natural foraging behavior of an ant colony, where ants find the shortest route from their nest to a food source using a simple communication mechanism. This mechanism is guided by a medium, i.e., pheromone, that is left by ants on the ground to mark a favorable path (i.e., a short path) and is used to guide other ants towards the target food source. In particular, ants perceive the presence of pheromones and tend to follow paths with a high pheromone concentration. This mechanism enables ants to transport food to their nest in a remarkably effective manner. Each ant randomly chooses a path for reaching food and leaves pheromones along the path. The pheromone concentration increases with the number of ants passing. A shorter path means that less time is spent on reaching food and returning, which results in more ants following this path in a certain time interval (that leads to a higher pheromone concentration). Fig. 4 illustrates the foraging behavior without an obstacle (Fig. 4a) and with an obstacle (Fig. 4b). If a path is blocked, there is an equal probability for an ant to choose the left or right path at first. As the right path (from *nest* to *food* passing point A) is shorter than the left path (from *nest* to *food* passing point B) that thus requires less travel time, it will end up with a higher pheromone concentration.



(a) Without an obstacle (b) With an obstacle

Fig. 4. Foraging behaviors of ants.

At a given point, when an ant has to choose among paths, it will choose the one with the highest pheromone concentration with higher probability. As a result, a high pheromone concentration is correlated with a short path. For example, as shown in Fig. 4b, more ants prefer the right path because of its higher pheromone concentration. Finally, all the ants will follow a shortest path. This colony-level behavior, which is based on the exploitation of positive feedback, is used by ants to find the shortest path between a food source and their nest.

4.3.2 Applying ACO to PTA

The PTA problem is to find an assignment with the maximum profit from a set of feasible task assignments. The original ACO [12] was applied to the classical Traveling Salesman Problem (TSP). In TSP, given a set of n towns, a traveling salesman aims to find a minimal length closed tour that visits each town once. The classic TSP is commonly used to formulate the problem of finding a path of the minimum cost [41]. However, the problem setting differs substantially from ours, and our problem cannot be directly converted to a classical TSP problem since our problem involves multiple workers (rather than only one traveling salesman) and there are dependencies among workers, i.e., the workers share all tasks. Therefore, the original algorithm does not solve our problem. In PTA, according to the reward pricing model in Section 3, workers with low travel times from their current location to a task's location can contribute more workload to completing the task, which is more likely to result in a higher or the maximum reward for the task. Due to the assumption that the profit of an SC

platform is proportional to the reward (cf. Definition 4), higher rewards for tasks imply a higher profit for the SC platform. To make the original ACO algorithm applicable to our problem, we consider a task-worker pair (s, w) as a path so that the distance between s and w is the path distance, while the original ACO considers the path between two towns.

When applying ACO to the PTA problem, a number of artificial ants are assigned to building solutions (corresponding to task assignments) to the PTA problem by exchanging information on the quality of the solutions via pheromones. ACO aims to find a solution with a set of short paths (i.e., a set of task-worker pairs with short distances) for the ants. However, in our problem, a task can be assigned to multiple workers but a worker can only perform one task at any time instance, whereas the original ACO solves the TSP problem where each town must be visited once and the traveling salesman can visit multiple towns. We improve the original ACO algorithm by considering the worker-task constraints.

Algorithm 5. ACO-based Task Assignment

Input: W, S, M
output: A, $A.P$
1: $A \leftarrow \emptyset$
2: Initialization of pheromone table T and heuristic information table E
3: **for** each task $s \in S$ **do**
4: Compute the reachable worker set $RW(s)$ for s;
5: Sort tasks in S according to $\frac{s.maxR}{s.wl}$ descendingly
6: $k \leftarrow 1$;
7: **repeat**
8: **foreach** $m \in M$ **do**
9: $A_m \leftarrow$ SolutionC$(S, \mathbb{RW}(S), T, E, m)$;
10: **if** $A.P < A_m.P$ **then**
11: $A \leftarrow A_m$
12: Update T;
13: $k \leftarrow k + 1$;
14: **until** $k > \xi$
15: /*ξ denotes the number threshold of iterations*/
16: **Return** A, $A.P$

Furthermore, we optimize the original ACO algorithm to obtain a higher platform profit. After the solution construction procedure using ACO, we continue to assign more workers to tasks that have the potential to obtain higher rewards. Specifically, given a task with potential to yield higher rewards, we keep assigning reachable workers to it until the maximum reward is achieved or no reachable workers exist.

Overview. Algorithm 5 offers an overview of the full ACO-based task assignment process. Given a worker set W, a task set S, and an ant set M, the task assignment A, pheromone table T, and heuristic information table E are initialized (lines 1–2). Then the reachable worker set $RW(s)$ for each task s is calculated (lines 3–4) while satisfying $\forall w \in RW(s)$, $t_{now} + t(w.l, s.l) \leq s.d \wedge d(w.l, s.l) \leq w.r$, where t_{now} denotes the current time, $t(a, b)$ denotes the travel time between location a and b, $s.d$ denotes the deadline of s, $d(a, b)$ denotes the distance between location a and b, and $w.r$ is the reachable distance of w. Tasks are sorted descendingly on their reward per unit of work, i.e., $\frac{s.maxR}{s.wl}$ (line 5).

Algorithm 5 performs a number of iterations (lines 7–14). In each iteration, a number of ants construct complete solutions by invoking the *SolutionC* procedure (line 9), and the task assignment solution with the highest profit is kept (lines 10–11). Then the pheromone table T is updated according to the above solution, to be covered in more detail later in this section. The iteration ends after a certain number of iterations, i.e., $k > \xi$, where ξ is the threshold number of iterations that can be specified by the SC platform (line 14). Finally, a good task assignment and its profit are obtained (line 16).

Solution Construction. Next we detail the *SolutionC* procedure for each ant m in Algorithm 6. The procedure takes five parameters: a task set (S), a reachable worker set for each task in S ($\mathbb{RW}(S) = \{RW(s)\}_{s \in S}$), a pheromone table ($T$), a heuristic information table (E), and an ant (m). After initialization (lines 1–2), Algorithm 6 assigns paths to each task until all the tasks are assigned, and computes an updated task assignment (lines 4–26). In each solution construction step, an ant selects the path to be visited according to a stochastic mechanism that is biased by pheromones (left by previous ants) and heuristic information. Specifically, for each task, we first calculate the selection probabilities of paths (i.e., $\mathbb{P}(s) = \{\mathcal{P}(s, w) | w \in RW'(s)\}$, where (s, w) denotes a path between s and w) associated with s based on Equation (7) (line 6). We then choose worker w from $RW'(s)$ based on $\mathbb{P}(s)$ and add w to the available task set of s when $RW'(s) \neq \emptyset$ (lines 7–9).

$$\mathcal{P}(s, w) = \begin{cases} \frac{[\tau(s,w)]^{\alpha} \cdot [\eta(s,w)]^{\beta}}{\sum_{w' \in RW'(s)} [\tau(s,w')]^{\alpha} \cdot [\eta(s,w')]^{\beta}} & \text{if } w \in RW'(s) \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

$$\eta(s, w) = \frac{1}{t(s.l, w.l) + 1}, \quad (8)$$

where $\mathcal{P}(s, w)$ denotes the selection probability of path (s, w), and $\tau(s, w)$ denotes the pheromone of path (s, w). Moreover, $\eta(s, w)$ denotes the heuristic information of path (s, w), which is the ratio between 1 and the travel time between w and s, as captured in Equation (8). In this setting, an ant tends to choose a path with less travel time, so that a closer worker is assigned to task s. Next, α ($\alpha \geq 0$) and β ($\beta \geq 0$) are parameters that control the contributions of $\tau(s, w)$ and $\eta(s, w)$ and that need to be fitted to the experimental data. $RW'(s)$ denotes the current reachable worker set of task s, from which the assigned workers have been removed compared to the original reachable worker set of task s (lines 10–11). Workers that are not reachable for task s are given selection probabilities of 0 for task s between s and them.

As seen in Algorithm 6, a task assignment solution contains a set of paths, each of which is denoted by a task-worker pair (e.g., (s, w) meaning that task s is assigned to worker w). A feasible solution must satisfy two constraints:

1) for each task-worker pair (s, w) in the solution, worker w is a reachable worker of task s, i.e., $w \in RW'(s)$, and
2) a task can be assigned to multiple workers but a worker can only perform one task at any time

instance, i.e., $\forall(s_i, w_k)$ and (s_j, w_g), $w_k \neq w_g$ (reflected in lines 11 and 23).

When the current task s cannot be finished before its deadline, its available worker set is set to the empty set, i.e., $AWS(s) \leftarrow \emptyset$ (lines 13–14).

Algorithm 6. SolutionC

Input: S, $\mathbb{RW}(S)$, T, E, m
output: A
1: $A \leftarrow \emptyset$
2: $\mathbb{RW}'(S) \leftarrow \mathbb{RW}(S)$
3: /* $\mathbb{RW}(S) = \{RW(s)\}_{s \in S}$ and $\mathbb{RW}'(S) = \{RW'(s)\}_{s \in S}$ */
4: **foreach** $s \in S$ **do**
5: $AWS(s) \leftarrow \emptyset$;
6: Compute the selection probabilities of paths (i.e., $\mathbb{P}(s) = \{\mathcal{P}(s, w) | w \in RW'(s)\}$, where (s, w) denotes a path between s and w) associated with s based on Equation (7);
7: **while** $RW'(s) \neq \emptyset$ **do**
8: Choose worker w from $RW'(s)$ based on $\mathbb{P}(s)$;
9: $AWS(s) \leftarrow AWS(s) \cup w$;
10: **if** task s is finished before its deadline by workers in $AWS(s)$ **then**
11: $\mathbb{RW}'(S) \leftarrow \mathbb{RW}'(S) - AWS(s)$;
12: break;
13: **if** task s cannot be finished before its deadline **then**
14: $AWS(s) \leftarrow \emptyset$;
15: **foreach** $s \in S$ **do**
16: **if** $AWS(s) \neq \emptyset$ and $P_{AWS(s)} < s.maxR$ **then**
17: **while** $RW'(s) \neq \emptyset$ **do**
18: **if** $t(w.l, s.l) < T(AWS(s))$ **then**
19: **if** $P_{AWS(s)} = s.maxR$ **then**
20: Break;
21: **else**
22: $AWS(s) \leftarrow AWS(s) \cup w$;
23: $\mathbb{RW}'(S) \leftarrow \mathbb{RW}'(S) - w$;
24: **else**
25: $RW'(s) \leftarrow RW'(s) - w$;
26: $A \leftarrow A \cup (s, AWS(s))$;
27: **Return** A;

To obtain a higher platform profit, we continue to assign more workers to tasks that have the potential to obtain higher rewards (lines 15–26). A task s has the potential to obtain higher rewards with more workers if reachable workers exist that are able to arrive at the location of s before its current completion time, i.e., $t(w.l, s.l) < T(AWS(s))$, where $t(w.l, s.l)$ denotes the travel time between location $w.l$ and $s.l$, and $T(AWS(s))$ denotes the task duration of s (i.e., the time from assignment to completion). Specifically, for a task s having the potential to yield higher rewards, we keep assigning reachable workers to it until the maximum reward is achieved or no reachable workers exist (lines 17–25). Then we update the task assignment solution A (line 26). Finally, a task assignment is returned (line 27). The time complexity of Algorithm 6 is $O(\mathcal{N} \cdot |maxRW|)$, where $\mathcal{N}$ denotes the number of tasks, and $|maxRW|$ denotes the maximum number of reachable workers across all tasks, i.e., $|maxRW| = max_{s \in S}|RW(s)|$. Accordingly, the time complexity of the ACO-based task assignment algorithm (cf. Algorithm 5) is $O(\max\{\mathcal{N} \cdot \mathbb{N}, \mathcal{N} \cdot \log \mathcal{N}, \xi \cdot |M| \cdot \mathcal{N} \cdot |maxRW|\})$, where $\mathbb{N}$ denotes the number of workers, ξ is the number of iterations, and $|M|$ denotes the number of ants.

Pheromone Updating. A key characteristic of Algorithm 5 is that in each iteration, the pheromone values are updated (line 12) by the ants that have built the solution in the iteration. The pheromone update in the kth ($k > 1$) iteration is described in the following equations. First, $\tau^k(s, w)$ denotes the pheromone on path (s, w) after k iterations and is computed as follows:

$$\tau^k(s, w) = (1 - \rho) \times \tau^{k-1}(s, w) + \sum_{m \in M} \Delta\tau_m^k(s, w), \tag{9}$$

where ρ is the evaporation rate of pheromone (from the previous to the current iteration), M denotes the ant set, and $\Delta\tau_m^k(s, w)$ denotes the quantity of pheromone given to path (s, w) by ant m in the kth iteration, which is calculated according to Equation (10).

$$\Delta\tau_m^k(s, w) = \begin{cases} P_{AWS(s)} - P_{AWS(s)-\{w\}} & \text{if } (s, AWS(s)) \in A_m^k \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

where $P_{AWS(s)} - P_{AWS(s)-\{w\}}$ denotes the profit increase by adding worker w to the available worker set (i.e., $AWS(s) - \{w\}$) of task s, and A_m^k denotes the task assignment of ant m in the kth iteration. The intuition is that if a path (s, w) can generate a higher profit for the task assignment, the path will contribute more profit to the solution. As a result, the ant tends to give more pheromones to this path, and other ants are more likely to select this path.

5 Experimental Evaluation

We evaluate the performance of the proposed methods on both real and synthetic datasets. The experimental setup is presented in Section 5.1, followed by a coverage of key experimental results in Section 5.2.

5.1 Experimental Setup

Datasets. We perform experiments on two datasets: *gMission* (GM) and *synthetic* (SYN). First, gMission is an open source SC dataset [2], where each task is associated with a location, a publication time, and a reward, and each worker is associated with a location and a reachable distance. As the gMission data is not associated with expected completion times, deadlines, and workloads and penalty rates of tasks, we uniformly generate these attributes from the ranges [10,20], $[s.e + 1, s.e + 20]$, [10, 20], and $[0, \frac{s.maxR}{s.d - s.e}]$ (to guarantee that the rewards obtained by workers are non-negative), respectively, where $s.d$ is the deadline of s and $s.maxR$ is the maximum reward of each task s. These settings are adjusted according to the actual data attributes, e.g., locations of workers and tasks, publication times of tasks, and reachable distances of workers. Values that are too small (including expected completion times and deadlines of tasks) or too large (including workloads and penalty rates of tasks) of the generated attributes cause most of the tasks to be unable to finish. In the other extreme, the use of too large expected completion times and deadlines of tasks or too large

TABLE 2
Parameter Settings

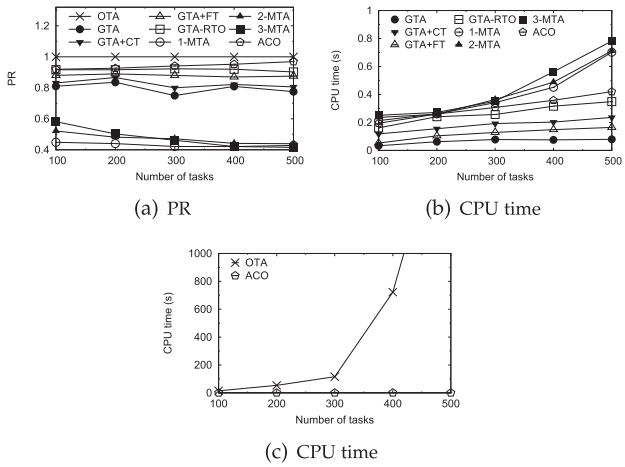
Parameters	Values
Number of tasks (GM), $\mathcal{N}$	100, 200, 300, 400, 500
Number of tasks (SYN), $\mathcal{N}$	1k, 3k, 5k, 7k, 9k
Number of workers (GM), $\mathbb{N}$	100, 200, 300, 400, 500
Number of workers (SYN), $\mathbb{N}$	1k, 3k, 5k, 7k, 9k
Abandoned probability of tasks in CT (GM&SYN), p_m^{CT}	0.1, 0.2, 0.3, 0.4
Reassigned probability of workers in FT (SYN), p_m^{FT}	0.2, 0.4, 0.6, 0.8
Early stop round in CT, FT and RTO (SYN), n	5, 10, 15, 20
The parameter measuring the contribution of pheromone on a path in ACO (SYN), α	0.4, 0.6, 0.8, 1.0, 1.2
The parameter measuring the contribution of heuristic information on a path in ACO (SYN), β	1.2, 1.4, 1.6, 1.8, 2.0
Evaporation rate of pheromone in ACO (SYN), ρ	0.1, 0.2, 0.3, 0.4, 0.5
Number of ants in ACO (SYN), $\lvert M\rvert$	3, 6, 9, 12, 15
Threshold number of iterations in ACO (SYN), ξ	5, 10, 15, 20, 25
Reachable distance of workers (SYN), r	0.2, 0.4, 0.6, 0.8, 1

workloads and penalty rates of tasks renders it impossible to observe the effects of the constraints imposed by these attributes. We set the maximum reward to $s.wl \cdot s.reward$ for each task s, where $s.wl$ denotes the workload of s and $s.reward$ is the reward of s in gMission. The speeds of workers are set to 1.

For the synthetic dataset, we generate locations of tasks and workers following a uniform distribution in a $2D$ space $[0, 30]^2$ based on observations from real datasets (e.g., gMission). The publication times of all tasks are set to 0, and each worker's reachable distance is set to 0.8. The expected completion times, deadlines, and workloads and penalty rates of tasks are uniformly generated from the ranges [5,20], $[s.e + 1, s.e + 15]$, [5,20], and $[0, \frac{s.maxR}{s.d - s.e}]$. The maximal reward of each task is set following a Gaussian distribution since it is influenced by complex variables in the real world. The speeds of workers in the synthetic dataset are set to 0.1.

Evaluation Methods. We evaluate performance of the following algorithms.

1) OTA: The Optimal Task Assignment (OTA) algorithm.
2) GTA: The basic Greedy Task Assignment (GTA) algorithm.
3) GTA + CT: GTA with Coarse Tuning.
4) GTA + FT: GTA with Fine Tuning.
5) GTA-RTO: GTA with Random Tuning Optimization (including coarse and fine tuning).
6) ACO: Ant Colony Optimization (ACO) based task assignment algorithm.
7) k-MTA: The Maximum Task Assignment (MTA) [15] based on a Minimum Cost Maximum Flow technique. In MTA, the weight between a worker (w) vertex and a task (s) vertex is set to $\frac{1}{d(w.l,s.l)}$, where $d(w.l, s.l)$ is the distance between w and s. The capacity between a task vertex and the fictitious destination vertex is set to k ($k = 1, 2, 3$), which means that at most k workers can be assigned to a task. MTA aims to maximize the number of task assignments by assigning workers to closer tasks with higher priorities.



(a) PR (b) CPU time

(c) CPU time

Fig. 5. Effect of $\mathcal{N}$ on the gMission dataset.

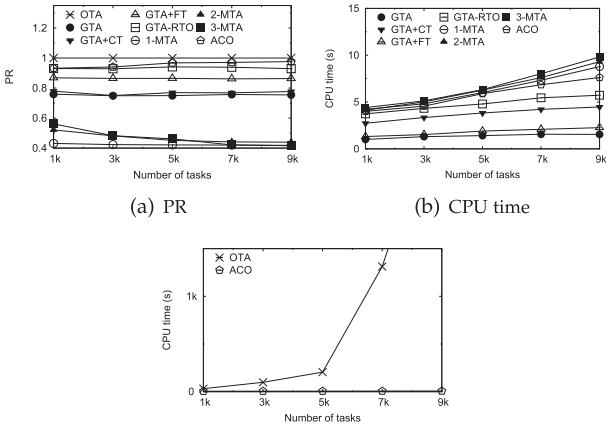
Metrics. Two metrics are compared among the above algorithms: *Profit-gaining Ratio* (*PR*, the ratio between the real and the optimal total platform profit) and *CPU time* (for finding the task assignment).

Table 2 shows the parameter settings, where the default values are underlined. All the experiments are conducted on an Intel(R) Core(TM) i7 − 10700 CPU @ 2.90GHz with 32.0 GB RAM.

5.2 Experimental Results

5.2.1 Scalability

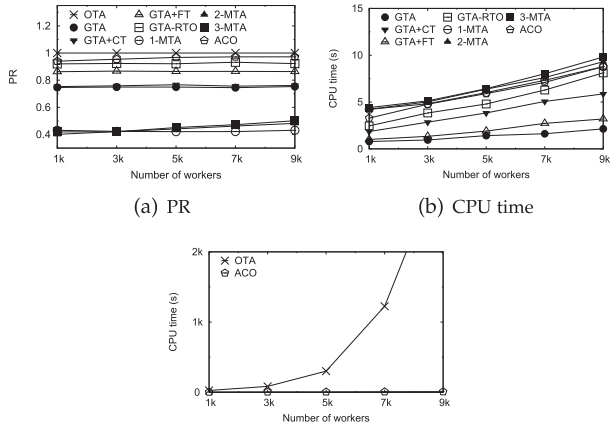
Effect of $\mathcal{N}$. We first study the effect of number of tasks $\mathcal{N}$. From Figs. 5a and 6a, we can see that OTA generates the highest profit-gaining ratio (i.e., PR = 1), followed by ACO, GTA-RTO, GTA+FT, GTA+CT, GTA, and k-MTA. ACO can achieve up to 97.60% of the overall profit of OTA, and it can improve the overall profit by up to 9.05% compared with GTA-RTO, which shows that ACO is superior in terms of obtaining profit. GTA-RTO is able to achieve at most 89.50% of the overall profit of OTA, which improves the total profit by up to 25.56% over GTA. We also notice that the profits generated by the k-MTA methods decline with increasing $\mathcal{N}$. This is so because the k-MTA methods only aim to maximize the number of task assignments (i.e., the number of matched task-worker pairs) while ignoring the completion of tasks. For example, a task that is assigned to at most k workers may not be finished by these workers, and thus the reward of this task is 0, and there is no contribution to the platform profit. Furthermore, with increasing $\mathcal{N}$, the k-MTA methods are more likely to assign fewer workers to a task, which results in more unfinished tasks and lower profits. This also explains why the k-MTA methods perform worse when k is smaller: each task is assigned to fewer workers with smaller k, leading to more unfinished tasks. In Figs. 5b, 5c, 5b, and 5c, although the CPU time of all



(a) PR (b) CPU time

(c) CPU time

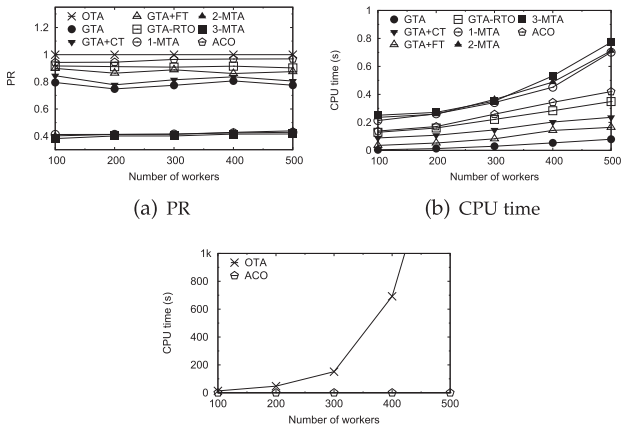
Fig. 6. Effect of $\mathcal{N}$ on the synthetic dataset.



(a) PR (b) CPU time

(c) CPU time

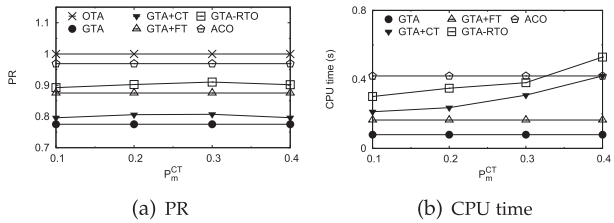
Fig. 8. Effect of $\mathbb{N}$ on the synthetic dataset.



(a) PR (b) CPU time

(c) CPU time

Fig. 7. Effect of $\mathbb{N}$ on the gMission dataset.



(a) PR (b) CPU time

Fig. 9. Effect of p_m^{CT} on the gMssion dataset.

methods increases with $\mathcal{N}$, the GTA-based approaches (including GTA, GTA+FT, GTA+CT, and GTA-RTO) perform better than the MTA-based approaches. ACO is slower than the GTA-based approaches, but it can improve the platform profit, which is crucial. OTA deteriorates much faster and cannot even return a result within a tolerable time duration when $\mathcal{N} > 400$ on the gMission dataset and when $\mathcal{N} > 7000$ on the synthetic dataset.

Effect of $\mathbb{N}$. Next we study the effect of $\mathbb{N}$ on the performance of the algorithms. Figs. 7a and 8a show that ACO performs better than all the GTA-based and MTA-based algorithms in terms of PR when varying $\mathbb{N}$. From Figs. 7b, 7c, 8b, and 8c, we can see that OTA is much slower than other methods. Although ACO is more time-consuming than the GTA-based algorithms, its CPU time is able, and it is able to achieve a good trade-off between effectiveness (second to OTA) and efficiency (second to GTA-based algorithms).

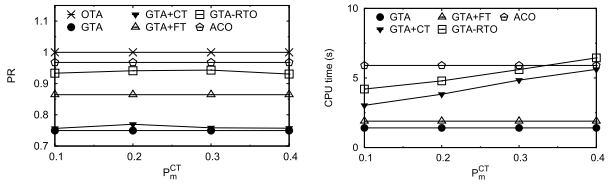
5.2.2 Performance of GTA-Based Algorithms

We proceed to evaluate the performance of the GTA-based algorithms for different parameter settings. We also include OTA and ACO for comparison. To save space, we do not report the CPU time of OTA, which is uncompetitive.

Effect of p_m^{CT}. In the coarse tuning, parameter p_m^{CT} represents the basic probability of abandoning a task. Note that parameters p_t^{CT} and p_r^{CT} are both set to $\frac{1-p_m^{CT}}{2}$. In Figs. 9a and 10a, GTA-RTO achieves the highest profit among the GTA-based methods, especially when p_m^{CT} is set to 0.3, which indicates that too conservative and too aggressive strategies are not suitable. In Figs. 9b and 10b, we see that the CPU time of GTA+CT and GTA-RTO increase when p_m^{CT} increases since more radical exploration strategies are more likely to reassign more workers when adopting the coarse tuning strategy. Further the performance of GTA, GTA+FT, OTA, and ACO stay stable in Figs. 9 and 10 since only strategies with coarse tuning are affected by variations in p_m^{CT}. In subsequent experiments, we omit results for the GM dataset, as these are similar to those obtained for the SYN dataset.

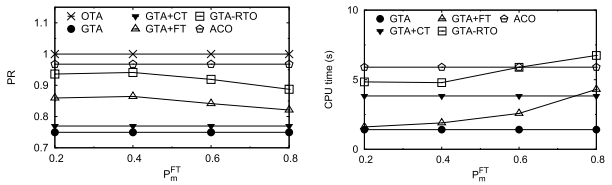
Effect of p_m^{FT}. We also evaluate the effect of varying p_m^{FT}, the basic probability parameter in fine tuning. Parameter p_t^{FT} is set to $1 - p_m^{FT}$. Fig. 11a shows that when compared with the GTA-based approaches, GTA-RTO can obtain higher profit. The CPU time of the methods with fine tuning (i.e., GTA-FT and GTA-RTO) increases when increasing p_m^{FT} (see Fig. 11b) for the same reason as p_m^{CT} increases, namely that a more radical exploration tends to generate more reassigned workers, thus incurring more CPU time for task assignment. Moreover, as only the fine tuning process is affected when varying p_m^{FT}, the methods without fine tuning (OTA, GTA, GTA+CT, and ACO) are unaffected, as shown in Fig. 11.

Effect of n. Next, we study the effect of varying n, called early stop round, that affects the termination condition in the GTA-based methods (except for GTA). In other words, GTA+CT, GTA+FT, and GTA-RTO keep searching until the nth iteration. OTA, GTA, and ACO are unaffected by n. From Fig. 12a, we can see that the profit of the GTA-based



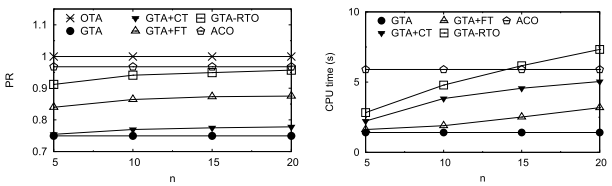
(a) PR (b) CPU time

Fig. 10. Effect of p_m^{CT} on the synthetic dataset.



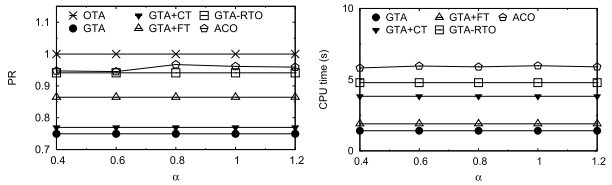
(a) PR (b) CPU time

Fig. 11. Effect of p_m^{FT} on the synthetic dataset.



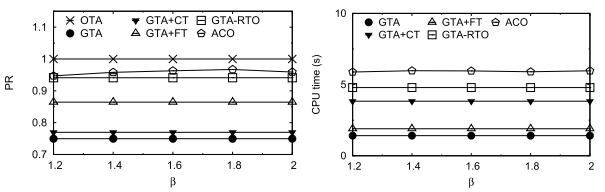
(a) PR (b) CPU time

Fig. 12. Effect of n on the synthetic dataset.



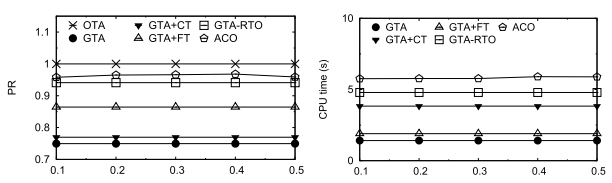
(a) PR (b) CPU time

Fig. 13. Effect of α on the synthetic dataset.



(a) PR (b) CPU time

Fig. 14. Effect of β on the synthetic dataset.



(a) PR (b) CPU time

Fig. 15. Effect of ρ on the synthetic dataset.

methods (except GTA) increase with increasing n. Specifically, a larger n leads to a higher possibility of finding a task assignment with more profit. Obviously, the CPU time of these methods increase in Fig. 12b when n increases, since a higher n means more iterations in the search process.

5.2.3 Performance of ACO

We next evaluate the performance of ACO across varying parameter settings in order to determine settings that work well for the data used. Moreover, we include OTA, GTA, GTA+CT, GTA+FT, and GTA-RTO as baseline algorithms.

Effect of α. Obviously, OTA, GTA, GTA+CT, GTA+FT and GTA-RTO are unaffected by variations in parameters that are specific to ACO, i.e., they are unaffected by α (measuring the contribution of pheromone on a path), as can be seen in Fig. 13. In Fig. 13a, PR first increases when increasing α. After peaking at $\alpha = 0.8$, PR drops steadily. It is worth noting that the profit obtained by ACO always exceeds those of the other algorithms (except OTA) for all α, demonstrating the superiority of ACO. In terms of efficiency, Fig. 13b shows that the CPU time of all the algorithms remain stable regardless of α.

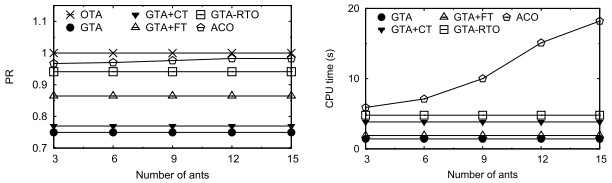
Effect of β. Next we study the effect of varying β that controls the contribution of heuristic information on a path in ACO. Fig. 14 shows that β has a similar effect on PR and CPU time as α has. Specifically, the PR of ACO increases as β is increased from 1.2 to 1.8, and then the PR drops slowly when $\beta > 1.8$ (see Fig. 14a). The CPU time of all approaches remain unchanged when varying β, shown in Fig. 14b.

Effect of ρ. We study the effect of varying ρ, the evaporation rate of pheromone deposited by previous ants, varying it from 0.1 to 0.5. As we can see in Fig. 15a, ACO achieves the highest PR when $\rho = 0.4$. Although ACO is the most time-consuming among all the algorithms (see Fig. 15b), it also outperforms the others in terms of PR.

Effect of $|M|$. Fig. 16 shows that both the PR and CPU time of ACO exhibit an increasing trend when the number of ants $|M|$ increases. More ants means that ants will follow more paths, which is more likely to result in a task assignment with higher profit. This explains the increasing trend in Fig. 16a. In addition, the CPU time of ACO increases since more paths need to be searched, as can be seen in Fig. 16b.

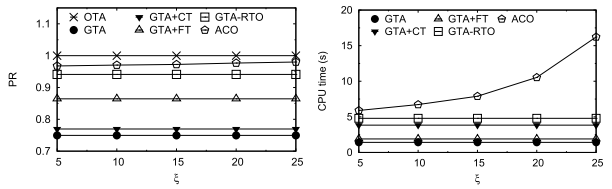
Effect of ξ. As expected, the profit of ACO increases gradually as ξ (threshold number of iterations in ACO) is increased—see in Fig. 17a. This is so because with more iterations, each ant tends to have more chances to choose a shorter path (i.e., a task assignment with lower travel times for workers), which leads to a task assignment with more profit. ACO still performs better than the GTA-based methods in terms of PR. However, the CPU time of ACO also increases when increasing the number of iterations, as shown in Fig. 17b. To achieve a good balance between effectiveness and efficiency, we use $\xi = 5$ as the default value in the experiments.

Effect of r. Finally, we study the effect of r (reachable distance of workers). Figs. 18a and 18b show that ACO can achieve task assignments with higher profits than the competitors, at the cost of increased CPU time. However, the computational cost of ACO remains acceptable. As illustrated in Fig. 18a, the profits of ACO first increases and then decreases gradually. This may be due to the fact that when r is first increased, each worker has more reachable tasks. As a result, each ant has more chances to choose a shorter path (i.e., a task assignment with lower travel time), which leads



(a) PR (b) CPU time

Fig. 16. Effect of $|M|$ on the synthetic dataset.



(a) PR (b) CPU time

Fig. 17. Effect of ξ on the synthetic dataset.

to task assignments with increased profits. When r reaches a certain level (i.e., $r \geq 0.4$), a task with a higher $\frac{s.maxR}{s.wl}$ value is more likely to be assigned to more workers since ACO prioritizes tasks with a higher reward per unit of work and each task has more available workers to choose among. This situation leads to more unassigned tasks with lower $\frac{s.maxR}{s.wl}$ values, thus decreasing the total profits. These profits of the GTA-related methods always decrease with growing r because they prioritize tasks with higher $\frac{s.maxR}{s.wl}$ values, leaving more tasks with lower $\frac{s.maxR}{s.wl}$ values unassigned. In Fig. 18b, the CPU time of ACO increases as r increases since each ant has to check more reachable workers for each task when r grows. In contrast, the CPU times of the GTA-related methods decline with growing r. The reason is that these methods end the assignment iterations early, when all available workers are assigned, which compensates for the CPU time needed for searching among more available workers.

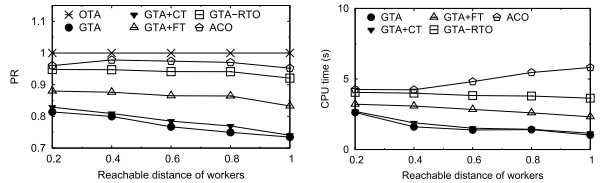
Summary of the Empirical Study. The findings of the empirical study can be summarized as follows:

1) OTA achieves the maximum profit but at the cost of very high CPU time;
2) GTA-RTO performs better than the other GTA-based methods in terms of profit, demonstrating the superiority of random tuning (including coarse and fine tuning) optimization;
3) ACO achieves a better balance between effectiveness (second to OTA) and efficiency (second to the GTA-based methods).

6 Related Work

6.1 Spatial Crowdsourcing

Crowdsourcing is a computing paradigm, where humans actively or passively participate in the process of computing, especially in the context of tasks that are intrinsically easier for humans to complete than for computers [34]. Many successful crowdsourcing platforms exist, e.g., Amazon Mechanical Turk (MTurk) and Wikipedia. Along with the ubiquity of GPS-equipped, networked devices like smart phones, a new class of crowdsourcing, called Spatial



(a) PR (b) CPU time

Fig. 18. Effect of r on the synthetic dataset.

Crowdsourcing (SC), has drawn increasing attention in both academia and industry. With SC, requesters can issue spatial tasks (e.g., monitoring traffic conditions) to SC servers that then assign these tasks to workers (called *task assignment*). The tasks are spatial because the workers must move to specified locations to complete the tasks. As one of the main enablers for the orchestration of location-based tasks, research on task assignment has gained considerable attention; consequently, many task assignment techniques have been proposed for different application scenarios [13], [22], [36], [39], [40], [45], [47]. Depending on how tasks are assigned to workers, SC can be classified into Server Assigned Tasks (SAT) mode and Worker Selected Tasks (WST) mode [15]. Most studies assume the SAT mode, where an SC server takes charge of the task assignment. We also adopt this mode. In SAT mode, the server assigns each task to nearby workers based on system optimization goals such as maximizing the number of assigned tasks after collecting all the locations of workers [7], [9], [15], [16], [19], [20], [44], maximizing the diversity score of assignments [5], maximizing the coverage of required skills of workers [3], or maximizing a global assignment quality score based on prediction [4]. For example, Kazemi and Shahabi [15] formulate SC as a matching problem between workers and tasks. They aim to maximize the total number of assigned tasks while conforming to workers' constraints, based on the assumption that the server has global knowledge of tasks and workers at each time instance. Tong et al. [32] propose a two-sided online micro-task assignment framework for spatial crowdsourcing that includes a TGOA-Greedy and a TGOA-OP algorithm. It differs from our work in terms of its task definition and problem setting. First, it defines a task as a micro-task, the processing time of which can be ignored. In contrast, we assign tasks with different workloads, which require certain amounts of time to complete. Second, the framework works in single-task assignment mode [15] and assigns only a single worker to a task. In contrast, we are able to assign multiple workers to a task to guarantee that the workload of a task can be completed. Due to these differences, their algorithms do not solve our problem.

In contrast, in WST mode, the server publishes spatial tasks online, and workers select tasks without coordination with the server [8], [9]. Deng et al. [8] formulate SC as a scheduling problem, reducing it to a specialized *Traveling Salesman Problem*. The authors propose exact and approximation algorithms to find a schedule that maximizes the number of tasks that can be completed by a worker, where travel costs of workers and expiration times of tasks are taken into consideration. Although task assignment has been the subject of many studies, open problems remain, including how to optimize the profit for an SC platform.

6.2 Profit-Aware Spatial Crowdsourcing

For an SC platform, an essential characteristic is its inherent cost effectiveness, as a platform aims to maximize profit during task assignment. However, many existing studies assume that an SC platform assigns tasks to mobile workers voluntarily for purposes of entertainment [1], information [14], or altruism [6], without considering the profit of the platform itself. In the settings of these studies, it is often difficult to engineer non-monetary incentive schemes for tedious and repetitive work [26], and they do not address the need for an SC platform to make a profit. Further, workers may not be willing to complete assigned tasks without payment because completing the tasks may incur participation costs (e.g., time and cost of travel and mobile device battery energy cost for sensing, data processing, and transmission).

Several profit-based task assignment mechanisms have been developed in crowdsourcing systems. For instance, in order to maximize the profit of a crowdsourcing system, Koutsopoulos [17] designs an optimal incentive mechanism through an optimal reverse auction with multiple winners. A key drawback is that the task allocation is time-consuming. Yang et al. [38] design incentive mechanisms for mobile crowdsourcing systems while taking the profit of the platform into consideration. They develop two system models (a platform-centric model and a user-centric model), but profit is sacrificed for computational efficiency. Shah-Mansouri et al. [25] design a profit maximizing truthful auction solution for mobile crowdsourcing systems, where the platform acts as an auctioneer and mobile workers act as sellers that submit bids to the platform. However, when assigning tasks, the above studies do not consider spatio-temporal information (e.g., location, mobility, and associated contexts), which plays a crucial role in SC. Furthermore, they focus on improving the total profit for an SC platform by providing different incentive mechanisms instead of targeting the task assignment process. In previous work [37], we formulate a profit-driven task assignment problem for SC, and we provide exact and greedy solutions. In the present study, we go further in this direction and adopt an ant colony optimization based approach to solve the profit-driven task assignment problem, which aims to achieve a better trade-off between effectiveness and efficiency.

7 Conclusion and Future Work

We study a novel SC problem, called Profit-driven Task Assignment (PTA), that aims to find a task assignment that maximizes the profit of an SC platform. In order to achieve high effectiveness and efficiency, we address key challenges by designing a reward pricing model to quantify the relationship between the task's reward and its completion time, and we develop exact and greedy algorithms to assign tasks. Furthermore, considering that an SC platform is likely to pursue high overall profits even at the cost of sacrificing some efficiency, we design a heuristic task assignment approach based on ant colony optimization, which achieves a better effectiveness-efficiency trade-off. An extensive empirical study based on real and synthetic data confirms the superiority of the paper's proposal. One future research direction is to consider the privacy leakage problem and design effective privacy protection strategies in task assignment.

Acknowledgments

We thank Guanfeng Liu, Jiajie Xu, and Min Zhang for their contributions to the previous conference version.

References

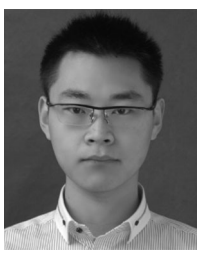
[1] L. V. Ahn, "Games with a purpose," *Computer*, vol. 39, no. 6, pp. 92–94, 2006.
[2] Z. Chen et al., "gMission: A general spatial crowdsourcing platform," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1629–1632, 2014.
[3] P. Cheng, X. Lian, L. Chen, and J. Han, "Task assignment on multi-skill oriented spatial crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 8, pp. 2201–2215, Aug. 2016.
[4] P. Cheng, X. Lian, L. Chen, and C. Shahabi, "Prediction-based task assignment in spatial crowdsourcing," in *Proc. IEEE Int. Conf. Data Eng.*, 2017, pp. 997–1008.
[5] P. Cheng et al., "Reliable diversity-based spatial crowdsourcing by moving workers," *Proc. VLDB Endowment*, vol. 8, no. 10, pp. 1022–1033, 2015.
[6] S. Cooper et al., "Analysis of social gameplay macros in the foldit cookbook," in *Proc. Int. Conf. Found. Digit. Games*, 2011, pp. 9–14.
[7] Y. Cui, L. Deng, Y. Zhao, B. Yao, V. W. Zheng, and K. Zheng, "Hidden POI ranking with spatial crowdsourcing," in *Proc. Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 814–824.
[8] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2013, pp. 324–333.
[9] D. Deng, C. Shahabi, and L. Zhu, "Task matching and scheduling for multiple workers in spatial crowdsourcing," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2015, pp. 1–10.
[10] W. Deng, J. Xu, and H. Zhao, "An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem," *IEEE Access*, vol. 7, pp. 20281–20292, 2019.
[11] W. Deng, H. Zhao, L. Zou, G. Li, X. Yang, and D. Wu, "A novel collaborative optimization algorithm in solving complex optimization problems," *Soft Comput.*, vol. 21, no. 15, pp. 4387–4398, 2017.
[12] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B*, vol. 26, no. 1, pp. 29–41, Feb. 1996.
[13] S. R. B. Gummidi, X. Xie, and T. B. Pedersen, "A survey of spatial crowdsourcing," *ACM Trans. Database Syst.*, vol. 44, no. 2, pp. 1–46, 2019.
[14] S. Jain, Y. Chen, and D. C. Parkes, "Designing incentives for online question and answer forums," in *Proc. ACM Conf. Electron. Commerce*, 2009, pp. 129–138.
[15] L. Kazemi and C. Shahabi, "GeoCrowd: Enabling query answering with spatial crowdsourcing," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2012, pp. 189–198.
[16] L. Kazemi, C. Shahabi, and L. Chen, "GeoTruCrowd: Trustworthy query answering with spatial crowdsourcing," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2013, pp. 314–323.
[17] I. Koutsopoulos, "Optimal incentive-driven design of participatory sensing systems," in *Proc. IEEE Conf. Comput. Commun.*, 2013, pp. 1402–1410.
[18] Q. Li, L. Cai, H. Xu, and T. Meng, "Profit maximization in mobile crowdsourcing: A competitive analysis," *IEEE Access*, vol. 9, pp. 27827–27839, 2021.
[19] X. Li, Y. Zhao, J. Guo, and K. Zheng, "Group task assignment with social impact-based preference in spatial crowdsourcing," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2020, pp. 677–693.
[20] X. Li, Y. Zhao, X. Zhou, and K. Zheng, "Consensus-based group task assignment with social impact in spatial crowdsourcing," *Data Sci. Eng.*, vol. 5, no. 4, pp. 375–390, 2020.
[21] Y. Li, M. Yiu, and W. Xu, "Oriented online route recommendation for spatial crowdsourcing task workers," in *Proc. Int. Symp. Spatial Temporal Databases*, 2015, pp. 137–156.
[22] Y. Li, Y. Zhao, and K. Zheng, "Preference-aware group task assignment in spatial crowdsourcing: A mutual information-based approach," in *Proc. IEEE Int. Conf. Des. Mater.*, 2021, pp. 350–359.

[23] M. Mahi, Ö. K. Baykan, and H. Kodaz, "A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem," *Appl. Softw. Comput.*, vol. 30, pp. 484–490, 2015.
[24] S. Sarker, M. A. Razzaque, M. M. Hassan, A. Almogren, G. Fortino, and M. Zhou, "Optimal selection of crowdsourcing workers balancing their utilities and platform profit," *Internet Things J.*, vol. 6, no. 5, pp. 8602–8614, 2019.
[25] H. Shah-Mansouri and V. W. S. Wong, "Profit maximization in mobile crowdsourcing: A truthful auction mechanism," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 3216–3221.
[26] Y. Singer and M. Mittal, "Pricing mechanisms for crowdsourcing markets," in *Proc. Int. World Wide Web Conf.*, 2013, pp. 1157–1166.
[27] T. Song et al., "Trichromatic online matching in real-time spatial crowdsourcing," in *Proc. IEEE Int. Conf. Data Eng.*, 2017, pp. 1009–1020.
[28] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1053–1064, 2016.
[29] Y. Tong, J. She, B. Ding, and L. Wang, "Online mobile micro-task allocation in spatial crowdsourcing," in *Proc. IEEE Int. Conf. Data Eng.*, 2016, pp. 49–60.
[30] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye, "Dynamic pricing in spatial crowdsourcing: A matching-based approach," in *Proc. Int. Conf. Manage. Data*, 2018, pp. 773–788.
[31] Y. Tong et al., "Flexible online task assignment in real-time spatial data," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1334–1345, 2017.
[32] Y. Tong, Y. Zeng, B. Ding, L. Wang, and L. Chen, "Two-sided online micro-task assignment in spatial crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 5, pp. 2295–2309, May 2021.
[33] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *Proc. VLDB Endowment*, vol. 11, no. 11, pp. 1633–1646, 2018.
[34] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi, "Spatial crowdsourcing: A survey," *Proc. VLDB Endowment*, vol. 29, no. 1, pp. 217–250, 2020.
[35] V. V. Vazirani, *Approximation Algorithms*. Berlin, Germany: Springer, 2013.
[36] Z. Wang, Y. Zhao, X. Chen, and K. Zheng, "Task assignment with worker churn prediction in spatial crowdsourcing," in *Proc. Conf. Inf. Knowl. Manage.*, 2021, pp. 2070–2079.
[37] J. Xia, Y. Zhao, G. Liu, J. Xu, M. Zhang, and K. Zheng, "Profit-driven task assignment in spatial crowdsourcing," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 1914–1920.
[38] D. Yang, G. Xue, F. Xi, and T. Jian, "Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 173–184.
[39] G. Ye, Y. Zhao, X. Chen, and K. Zheng, "Task allocation with geographic partition in spatial crowdsourcing," in *Proc. Conf. Inf. Knowl. Manage.*, 2021, pp. 2404–2413.
[40] Y. Zhao, J. Guo, X. Chen, J. Hao, X. Zhou, and K. Zheng, "Coalition-based task assignment in spatial crowdsourcing," in *Proc. IEEE Int. Conf. Data Eng.*, 2021, pp. 241–252.
[41] Y. Zhao and Q. Han, "Spatial crowdsourcing: Current state and future directions," *Commun. Mag.*, vol. 54, no. 7, pp. 102–107, 2016.
[42] Y. Zhao, Y. Li, Y. Wang, H. Su, and K. Zheng, "Destination-aware task assignment in spatial crowdsourcing," in *Proc. Conf. Inf. Knowl. Manage.*, 2017, pp. 297–306.
[43] Y. Zhao et al., "Preference-aware task assignment in spatial crowdsourcing," in *Proc. Conf. Assoc. Advance. Artif. Intell.*, 2019, pp. 80–87.
[44] Y. Zhao, K. Zheng, Y. Cui, H. Su, F. Zhu, and X. Zhou, "Predictive task assignment in spatial crowdsourcing: A data-driven approach," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 13–24.
[45] Y. Zhao, K. Zheng, J. Guo, B. Yang, T. B. Pedersen, and C. S. Jensen, "Fairness-aware task assignment in spatial crowdsourcing: Game-theoretic approaches," in *Proc. IEEE Int. Conf. Data Eng.*, 2021, pp. 265–276.
[46] Y. Zhao, K. Zheng, Y. Li, H. Su, J. Liu, and X. Zhou, "Destination-aware task assignment in spatial crowdsourcing: A worker decomposition approach," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 12, pp. 2336–2350, Dec. 2019.
[47] Y. Zhao, K. Zheng, H. Yin, G. Liu, J. Fang, and X. Zhou, "Preference-aware task assignment in spatial crowdsourcing: From individuals to groups," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 7, pp. 3461–3477, Jul. 2022.
[48] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, pp. 2687–2699, 2015.

Yan Zhao received the Doctoral degree in computer science from Soochow University in 2020. She is an assistant professor with Aalborg University. Her research interests include spatial database and trajectory computing.

Kai Zheng (Senior Member, IEEE) received the PhD degree in computer science from The University of Queensland in 2012. He is a professor of Computer Science with the University of Electronic Science and Technology of China. He has been working in the area of spatial-temporal databases, uncertain databases, social-media analysis, in-memory computing and blockchain technologies. He has published more than 100 papers in prestigious journals and conferences in data management field such as SIGMOD, ICDE, *VLDB Journal*, *ACM Transactions* and *IEEE Transactions*.

Yunchuan Li received the bachelor's degree in software engineering from Southwest Petroleum University, in 2019. He is currently working toward the master's degree in computer science and technology with the University of Electronic Science and Technology of China. His research interests include spatial database and anomaly detection.



Jinfu Xia received the master's degree in computer science from Soochow University in 2020. His research interests include spatial database and trajectory computing.

Bin Yang received the PhD degree in computer science from Fudan University, China. He is a professor with Aalborg University, Denmark. He was previously with Aarhus University, Denmark and with MaxPlanck-Institut fur Informatik, Germany. His research interests include machine learning and data management.

Torben Bach Pedersen (Senior Member, IEEE) is a professor with the Center for Data-Intensive Systems (Daisy), Aalborg University, Denmark. His research concerns data analytics for "Big Multidimensional Data" - the integration and analysis of large amounts of complex and highly dynamic multidimensional data. He is an ACM distinguished scientist and a member of the Danish Academy of Technical Sciences.

Rui Mao received the BS and MS degrees in computer science from the University of Science and Technology of China, in 1997 and 2000, respectively, and the MS degree in statistics and PhD degree in computer science from the University of Texas at Austin, in 2006 and 2007, respectively. After three years work with the Oracle USA Corporation, he joined Shenzhen University in 2010. He is now associate dean with the College of Computer Science and Software Engineering, Shenzhen University, and executive director with the Shenzhen Institute of Computing Sciences. His research mainly focuses on universal data management and analysis.

Christian S. Jensen (Fellow, IEEE) is a professor of computer science with Aalborg University, Denmark. He was a professor with Aarhus University from 2010 to 2013, and he was previously with Aalborg University for two decades. His research concerns data management and analytics, and its focus is on temporal and spatio-temporal data. He is a member of the Academia Europaea, the Royal Danish Academy of Sciences and Letters, and the Danish Academy of Technical Sciences. He has received several national and international awards for his research, most recently the IEEE TCDE impact award. He is a fellow of ACM.

Xiaofang Zhou (Fellow, IEEE) received the bachelor's and master's degrees in computer science from Nanjing University, in 1984 and 1987, respectively, and the PhD degree in computer science from the University of Queensland in 1994. He is the Otto Poon professor of Engineering and chair professor of Computer Science and Engineering with the Hong Kong University of Science and Technology. He is head with the Department of Computer Science and Engineering. His research is focused on finding effective and efficient solutions to managing integrating, and analyzing very large amounts of complex data for business and scientific applications. His research interests include spatial and multimedia databases, high performance query processing, web information systems, data mining, and data quality management.