Multi-Worker-Aware Task Planning in Real-Time Spatial Crowdsourcing

Qian Tao[1], Yuxiang Zeng[2], Zimu Zhou[3], Yongxin Tong[1]*, Lei Chen[2], and Ke Xu[1]

[1] SKLSDE Lab and BDBC, Beihang University, China
[1]{qiantao,yxtong,kexu}@buaa.edu.cn
[2] The Hong Kong University of Science and Technology, Hong Kong SAR, China
[2]{yzengal,leichen}@cse.ust.hk
[3] Laboratory TIK, ETH Zurich, Zurich, Switzerland
[3]zzhou@tik.ee.ethz.ch

Abstract. Spatial crowdsourcing emerges as a new computing paradigm with the development of mobile Internet and the ubiquity of mobile devices. The core of many real-world spatial crowdsourcing applications is to assign suitable tasks to proper workers in real time. Many works only assign a set of tasks to each worker without making the plan how to perform the assigned tasks. Others either make task plans only for a single worker or are unable to operate in real time. In this paper, we propose a new problem called the *Multi-Worker-Aware Task Planning (MWATP)* problem in the online scenario, in which we not only assign tasks to workers but also make plans for them, such that the total utility (revenue) is maximized. We prove that the offline version of MWATP problem is NP-hard, and no online algorithm has a constant competitive ratio on the MWATP problem. Two heuristic algorithms, called Delay-Planning and Fast-Planning, are proposed to solve the problem. Extensive experiments on synthetic and real datasets verify the effectiveness and efficiency of the two proposed algorithms.

Keywords: Spatial crowdsourcing, Task assignment, Task planning.

1 Introduction

The development of mobile devices has triggered the fast growing of spatial crowdsourcing. Unlike traditional crowdsourcing where workers perform tasks via webs [1], workers in spatial crowdsourcing need to physically go to the location of a task to perform it [2]. Spatial crowdsourcing extends traditional crowdsourcing to the physical world and has seen many applications in daily life [3][4]. For example, Waze[4] provides a dynamic traffic navigation by collecting the GPS information; Uber[5] offers an efficient real-time taxi-calling service; Gigwalk[6] performs location-based micro tasks via crowds, *etc.*

* Corresponding author.
[4] http://www.waze.com
[5] http://www.uber.com
[6] http://www.gigwalk.com

Table 1: Release Time, Expiration Time and Utility

Task/Worker	t_1	w_1	t_2	t_3	w_2	t_4	t_5
Release time	1	1	1.5	2	3	5	5.5
Expiration time	4	6.8	3	5	8.2	6.2	7
Utility (Revenue)	5	／	2	3	／	2	1

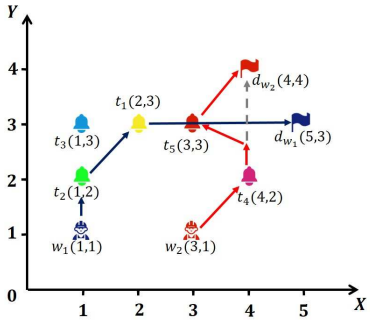


Fig. 1: Initial Locations of Workers and Tasks

One of the most important issues in spatial crowdsourcing research is how to assign tasks to proper workers [5][6][7][8]. Imagine the following scenario. Suppose Alice is off duty at 5:00 p.m. from her office, and she wants to perform some tasks passingly from Gigwalk on her way home. However, she has to reach home before 6:00 p.m. to have dinner with her family. Thus Alice wants to receive not only the guidance of which tasks to perform, but also a plan (order) to perform them. Every performed task contributes a revenue to the platform. When *multiple workers* raise such demands *in real time*, the platform faces a new problem in spatial crowdsourcing: *how to make plans of tasks for multiple workers in online scenario, such that the total revenue of the platform is maximized?*

We further illustrate our motivation using the following example.

Example 1. Suppose there are two workers $w_1 - w_2$ and five tasks $t_1 - t_5$ appearing on the platform, whose release and expiration times (in minutes) are shown in Table 1, and the locations are shown in Fig. 1. The "utility" of each task is also shown in Table 1, representing the revenue contributed to the platform when the task is performed. For ease of presentation, the coordinates in Fig. 1 have been transformed to the corresponding time. Workers and tasks can be observed after their release times, and tasks cannot be performed after their expiration times. At time 1, only the task t_1 and the worker w_1 are observed. Along a route $\langle(1,1),(1,2),(2,3),(5,3)\rangle$ (blue arrows in Fig. 1), w_1 performs $\{t_1, t_2\}$, and reaches his/her destination d_{w_1} at time 6.41 (we calculate the time accurately to 2 decimal places), which is earlier than w_1's expiration time. Since t_5 appears at time 5.5, w_1 cannot reach his/her destination d_{w_1} earlier than time $r_{t_5} + dis(l_{t_5}, d_{w_1}) = 7.5$ if he/she performs t_5. w_2 appears at time 3, but no task can be accomplished by w_2 right away. Suppose we let w_2 move to location $(4,2)$ and stay. At time 5, t_4 appears, and it can be accomplished by w_2. We then alter w_2's plan to be "accomplishing t_4 and moving to his/her destination"

(gray arrow in Fig. 1). However, at time 5.5, observing that t_5 appears and it can be accomplished by w_2 (w_2 is at location $(4, 2.5)$ right now), we redirect w_2 to perform t_5 before he/she moves to his/her final destination. w_2 can finally accomplish t_4 and t_5. Based on the above plans for w_1 and w_2, we obtain a total utility (revenue) of 10, which is the optimal planning in this instance.

Many works model the task assignment problem as an online bipartite graph matching and only assign a set of tasks to the workers without indicating an order to perform them [2][9][10]. Some pioneer works have explored task planning [11][12][13][14][15][16]. However, they either are designed for a single worker [11][13], or cannot handle the real-time (*i.e.*, online) scenario [12][14][15][16].

In this paper, we propose a new task assignment problem for real-time spatial crowdsourcing, called the *Multi-Worker-Aware Task Planning (MWATP)* problem. We attempt to not only assign a set of tasks to multiple workers, but also make plans for them, to maximize the total utility (revenue) contributed to the platform, in the *two-sided online* scenario (*i.e.* both workers and tasks appear on the platform dynamically). In summary, we make the following contributions.

- We formulate the Multi-Worker-Aware Task Planning *(MWATP)* problem, which assigns tasks and makes plans for multiple workers in online scenario, such that the total utility (revenue) is maximized. We prove that the offline MWATP problem is NP-hard, and any online algorithm for the online MWATP problem has no constant competitive ratio.
- We propose two heuristic algorithms, called Delay-Planning and Fast-Planning to solve the online MWATP problem.
- We conduct extensive experiments on both synthetic and real datasets. Evaluations verify the effectiveness and efficiency of our proposed algorithms.

The rest of the paper is organized as follows. We formally define the MWATP problem and prove its hardness in Sec. 2. Two heuristic algorithms are proposed in Sec. 3. We present the experimental evaluations in Sec. 4, review related work in Sec. 5, and finally conclude this work in Sec. 6.

2 The MWATP Problem

In this section, we first formally define the *Multi-Worker-Aware Task Planning (MWATP)* problem, and then prove its hardness.

2.1 Problem Definitions

This subsection presents the formal definition of the *Multi-Worker-Aware Task Planning* problem.

Definition 1 (Task). *A task t is a tuple $\langle l_t, r_t, e_t, u_t \rangle$, where l_t is the location which requires the worker to reach, r_t and e_t are the release time and expiration time of task t, and u_t is the utility (revenue) contributed to the platform if the task is accomplished. The task t can be observed only after its release time r_t, and it cannot be performed after its expiration time e_t. The time interval $[r_t, e_t]$ is called the* valid interval *of t.*

We assume the release time of a task is always no greater than its expiration time, *i.e.*, $r_t \leq e_t$ for $\forall t \in T$, because otherwise the task will never be finished. We also assume a task can only be performed by one worker. Besides, we use "utility" to represent the revenue of a task hereafter.

Definition 2 (Worker). *A worker w is a tuple $w = \langle s_w, d_w, c_w, r_w, e_w \rangle$, where w appears at his/her release time r_w with the initial location s_w, and needs to reach his/her destination d_w before the expiration time e_w. We use c_w to denote the current location of w at a certain time T^*. Specifically, c_w equals the initial location s_w when w appears on the platform. A worker can accomplish a task t if he/she can reach the location of t within the valid interval $[r_t, e_t]$, which will add a utility value of u_t for the platform.*

We model the locations on a metric space (M, dis), where M is a set of locations, and dis is a function, $dis : M \times M \rightarrow R$, and assume each worker can reach his/her destination before his/her expiration time, *i.e.*, $e_w \geq r_w + dis(s_w, d_w)$ for $\forall w \in W$. For simplicity, each worker is assumed to travel at the same constant speed. Consequently, a distance can be represented by a time period, and we will follow this rule whenever there is no ambiguity.

Definition 3 (Guidance). *A guidance for a worker w is a tuple $g = \langle t_g, DIR(l) \rangle$, which means at time t_g, worker w needs to head to the location l from his/her current location. Note that l can be the same as the worker's current location. In this case, we denote $DIR(l)$ as $STAY$.*

Definition 4 (Plan). *A plan for a worker w is a vector of guidance $p_w = \langle g_1, g_2, ..., g_{|p_w|} \rangle$, where $t_{g_i} < t_{g_{i+1}}$ for $i = 1, 2, ..., |p_w - 1|$. A plan is* valid *if w can reach his/her destination d_w before his/her expiration time e_w following the plan p_w. Given a set of tasks T, we further denote $AT(T, p_w)$ as the set of tasks that can be accomplished by w following p_w.*

Note that the plan p_w for a worker w can be updated by the platform when new tasks appear on the platform. However, the updated plan should always be valid. With a plan p_w, we can generate a route of the worker. Based on this route, we can check whether a task can be accomplished, *i.e.* a task $t \in T$ belongs to $AT(T, p_w)$, by checking whether w can reach l_t within t's valid interval $[r_t, e_t]$. Given a set of plans P for multiple workers on a set of tasks T, we define the set of accomplished tasks $AT(T, P) = \cup_{p \in P} AT(T, p)$.

Definition 5 (Online Multi-Worker-Aware Task Planning Problem). *Given a set of workers W and a set of tasks T, where workers and tasks arrive one by one according to their release times, the problem is to find a valid plan set P for W, such that the total utility of accomplished tasks,* i.e.*,*

$$U(T, P) = \sum_{t \in AT(T,P)} u_t \tag{1}$$

is maximized.

In this paper we mainly study the online MWATP problem. If not explicitly specified, we will use "MWATP" to refer to the online MWATP problem.

Example 2. Assume the same settings as in Example 1. Then the plan for w_1 is $p_{w_1} = \langle\langle 1, DIR((1,2))\rangle, \langle 2, DIR((2,3))\rangle, \langle 3.41, DIR((5,3))\rangle\rangle$, which generates a route $\langle(1,1),(1,2),(2,3),(5,3)\rangle$. Similarly, the plan for w_2 is $p_{w_2} = \langle\langle 3, DIR((4,2))\rangle, \langle 4.41, STAY\rangle, \langle 5, DIR((4,4))\rangle, \langle 5.5, DIR((3,3))\rangle, \langle 6.62, DIR((4,4))\rangle\rangle$, which also generates a route $\langle(3,1),(4,2),(4,2.5),(3,3),(4,4)\rangle$. The set of tasks accomplished from $P = \{p_{w_1}, p_{w_2}\}$ is $AT(T,P) = AT(T,p_{w_1}) \cup AT(T,p_{w_2}) = \{t_1, t_2, t_4, t_5\}$.

2.2 Hardness of MWATP Problem

In this subsection, we first show that the offline MWATP problem is NP-hard, and then prove that no algorithm can achieve a constant competitive ratio on the MWATP problem.

Definition 6 (Offline Multi-Worker-Aware Task Planning Problem). *Given a set of workers W and a set of tasks T, where the release times of workers and tasks are known a priori, the problem is to decide a valid plan set P for W, such that the total utility of the accomplished tasks is maximized.*

Theorem 1. *The offline MWATP problem is NP-hard.*

Proof. We prove the NP-hardness of offline MWATP problem by reducing the orienteering problem [17] to it. The decision version of MWATP problem is to decide if there is a valid plan set P, such that total utility is no less than U. The decision version of orienteering problem is defined as follows. Given n nodes, where one is the start node s_1, one is the end node s_n, and each of the other $n-2$ nodes is associated with a score, the objective is to find a route of nodes starting from s_1 and ending at s_n, such that the total score is no less than S, with a time constraint T_{MAX}. For an instance I of the orienteering problem, we map the start node s_1 to the worker's start location, the end node s_n to the worker's destination, the other $n-2$ nodes to $n-2$ tasks, and the decision threshold S to U. Let the release time and expiration time of each task and the worker be 0 and T_{MAX}, respectively. Now we get an instance I' of the offline MWATP problem. In I', a task can be performed at any time, as long as the worker can reach his/her destination on time. This means that as long as there is a route for the worker in I' achieving utility U, then there must be a route in the orienteering problem gaining the same scores, and vice versa. Since the decision version of orienteering problem is NP-complete, the optimization version of offline MWATP problem is NP-hard. □

Next we prove that for the MWATP problem, neither deterministic nor randomized online algorithm can yield a constant competitive ratio. Although a similar claim of a special case of the MWATP problem has been considered in [13], it neglects the proof on randomized algorithms.

Lemma 1. *No deterministic algorithm for the MWATP problem has a constant competitive ratio.*

Proof. The problem in [13] is a special case of the MWATP problem with a single worker. Since the problem in [13] does not have a deterministic algorithm with constant competitive ratio, the MWATP problem does not have a deterministic algorithm with constant competitive ratio either. □

Lemma 2. *No randomized algorithm for the MWATP problem has a constant competitive ratio.*

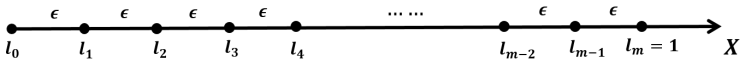


Fig. 2: An Instance that Randomized Algorithms Perform Bad

Proof. We prove the lemma by showing that the MWATP problem with exactly one worker does not have a constant competitive ratio. Consider an instance shown in Fig. 2. We omit the Y axis since tasks and workers appear on the X axis. l_0 is the origin with coordinate $(0, 0)$. Let m be an arbitrary positive integer and $\epsilon = \frac{1}{m}$. At time 1, with probability $\frac{1}{m}$, n tasks appear at location l_i with expiration time $1 + \frac{\epsilon}{2}$. All of the tasks have a utility value of 1. This yields a probability distribution $\mathcal{X}$ over the input of the tasks. At time 0, a worker w appear at l_0, with the destination $l_m = (1, 0)$ and expiration time 2. No matter where the n tasks appear, in the optimal solution w can wait at the location until tasks appear, and then go to the destination before his/her expiration time. Therefore the optimal result on $\mathcal{X}$ is $\mathbb{E}_{\mathcal{X}}[OPT] = n$. Now consider a generic deterministic online algorithm ALG. The worker at most reach one location of $l_1, l_2, ..., l_m$ before the tasks' expiration time, no matter where he/she is located at time 1. This means that the expectation of the utility value under the input distribution $\mathcal{X}$ is at most $\mathbb{E}_{\mathcal{X}}[ALG] \leq \frac{1}{m} \cdot n = n\epsilon$. This yields

$$\frac{\mathbb{E}_{\mathcal{X}}[ALG]}{\mathbb{E}_{\mathcal{X}}[OPT]} \leq \frac{n\epsilon}{n} = \epsilon \tag{2}$$

The ratio for any deterministic online algorithm becomes unbounded when ϵ is small enough. From Yao's Principle [18], no randomized algorithm for the MWATP problem can achieve a constant competitive ratio. □

Theorem 2. *No online algorithm, neither deterministic nor randomized, can achieve a constant competitive ratio on MWATP problem.*

Proof. The theorem is a direct result from Lemma 1 and 2. □

3 Solutions to MWATP Problem

Although no deterministic or randomized algorithms can achieve a constant competitive ratio, we propose two efficient heuristic algorithms, Delay-Planning and Fast-Planning, to solve the MWATP problem.

3.1 The Delay-Planning Algorithm

Main Idea. In the Delay-Planning algorithm, a worker neglects the new tasks while he/she is executing his/her current plan. Once the current plan is finished, the worker is assigned a new plan with the delayed (previously neglected) tasks.

Algorithm 1: BenefitGreedy

input : a worker w and a set of tasks T
output: worker w with new plan

```
1  Sort t in T according to BEN(w, t) in descending order;
2  end_loc ← c_w;
3  end_time ← the current time;
4  foreach task t ∈ T do
5      if end_time + dis(end_loc, l_t) ≤ e_t and
       end_time + dis(end_loc, l_t) + dis(l_t, d_w) ≤ e_w then
6          Append t to S_w;
7          end_time ← end_time + dis(end_loc, l_t);
8          end_loc ← l_t;
9          T ← T − t;
10 if no task is assigned to w then
11     Let w move toward to d_w;
```

Algorithm Details. We use a task pool to store the tasks that have not been assigned to workers. Whenever a new worker arrives or a worker finishes his/her last plan, the algorithm finds a new plan for the worker from the task pool.

We apply a succinct greedy function to make new plans for a worker by considering both the utility and the distance from the worker's current location to the task. *(i)* A task with a higher utility is preferred. *(ii)* A larger distance between the task and the worker leads to a higher risk of the expiration of the task. Combining these two considerations, we use the ratio between the utility and the distance from the worker, denoted by $BEN(w,t) = \frac{u_t}{dis(c_w, l_t)}$, to measure the *benefit* of a task. The function greedily chooses the next task with the largest benefit that can be accomplished on time by the worker.

Alg. 1 illustrates the procedure of the greedy function. In line 1, the tasks in the task pool are sorted according to their benefits from w. In lines 2-3, two variables *end_loc* and *end_time* are defined to represent the location and time when the worker finishes his/her current plan. For each task t, we judge if the worker can accomplish it and reach the destination on time if t is appended to the tail of S_w in line 5. Note that S_w represents the task sequence of w, as is shown in Alg. 2. If "yes", then the algorithm assigns t to w, updates *end_time* and *end_loc*, and removes t from the set T in lines 6-9.

The Delay-Planning algorithm is built upon the $BenefitGreedy$ function (see Alg. 2). In lines 1-2, we initialize a task pool $taskPool$, and a free worker set $freeWorkerSet$. In line 3, each worker $w \in W$ is associated with a task sequence S_w, *i.e.*, a plan. Whenever a task arrives ("true" judgement in line 5), we first attempt to assign it to the workers in $freeWorkerSet$ in lines 6-8. If fail, we add the task to $taskPool$ in lines 9-10. When a worker arrives, we assign tasks and update his/her plan from the task pool $taskPool$ for him/her in line 13. A worker who has just finished his/her current plan is regarded as a new worker in Delay-Planning (see lines 12-15).

Algorithm 2: Delay-Planning

input : A set of workers W, a set of tasks T
output: Plans for $w \in W$

```
1  taskPool ← ∅;
2  freeWorkerSet ← ∅;
3  Set S_w an empty task sequence for each w ∈ W;
4  for each new arrival request do
5      if the request is a task t then
6          if there exists a worker w' ∈ freeWorkerSet can accomplish t with
           largest BEN then
7              Append t to S_w';
8              freeWorkerSet ← freeWorkerSet − {w'};
9          else
10             taskPool ← taskPool ∪ {t};
11     else
12         // Denote the arrival worker by w.
13         BenefitGreedy(w, taskPool);
14         if there is no task appended to S_w then
15             freeWorkerSet ← freeWorkerSet ∪ {w};
```

Example 3. Back to our running example in Example 1. When w_1 appears, there is one task, t_1, in the task pool. We then invoke $BenefitGreedy(.,.)$ to make a plan for w_1. Since w_1 can reach l_{t_1} before e_{t_1}, and reach his/her destination on time, w_1's new plan is to accomplish $\{t_1\}$. Then t_2 and t_3 appear at time 1.5 and 2 but there is no worker in $freeWorkerSet$. So they are added to $taskPool$. At time 3, w_2 appears. Now $taskPool$ is $\{t_3\}$, because t_2 has expired. However, w_2 cannot accomplish t_3 before the expiration time of t_3. Thus w_2 directly moves to d_{w_2}. At time 3.24, w_1 finishes the last task sequence $\langle t_1 \rangle$ and now $taskPool = \{t_3\}$. w_1 cannot accomplish t_3 before the expiration time of w_1. Hence w_1 directly moves to d_{w_1}. At time 5, t_4 appears. Currently $freeWorkerSet = \{w_1, w_2\}$. The locations of w_1 and w_2 are $(3.76, 3)$ and $(3.63, 2.90)$, respectively. t_4 is assigned to w_2. We cannot choose w_1 because of w_1's expiration time. At time 5.5, t_5 appears. Neither w_1 nor w_2 can accomplish it. Note that at this time, w_2 is still accomplishing his/her current task sequence $\langle t_4 \rangle$. Finally, $S_{w_1} = \langle t_1 \rangle$ and $S_{w_2} = \langle t_4 \rangle$. The total utility is $u_{t_1} + u_{t_4} = 7$.

Time Complexity. We apply the amortized analysis to analyze the complexity of Alg. 2. Assume n and m are the number of workers and tasks, respectively. First, the time complexity for calling Alg. 1 is $O(m \log m)$. In Alg. 2, the time complexity of lines 6-8 is $O(n)$, and they are executed at most m times. The time complexity of lines 5-10 is $O(mn)$. In lines 11-15, a worker may become a request more than one time. However, this happens only when he/she appears for the first time or just accomplishes a plan, which means that lines 11-15 are executed at most $O(m + n)$ times (n for appearing and m for accomplishing a plan). The total complexity of lines 11-15 is $O((m + n)m \log m)$. Combing these two parts, the time complexity of Delay-Planning is $O((m + n)m \log m)$.

3.2 The Fast-Planning Algorithm

The Delay-Planning algorithm defers the processing of tasks for a certain time, which potentially leads to the expiration of some tasks. Thus we further propose the Fast-Planning algorithm to fasten the process of making new plans, and therefore, potentially increase the total utility.

Main Idea. Whenever a task appears, the Fast-Planning algorithm immediately assigns the task to a worker and makes a new plan for the worker. To make the new plan efficiently, the algorithm only attempts to combine the new task with the current plan, rather than going through all possible permutations.

Algorithm Details. Alg. 3 illustrates the procedure of the Fast-Planning algorithm. In line 1, we initialize two sets, $aWorkerSet$ and $freeTaskSet$, representing the available worker set and the unassigned task set, respectively. Whenever a worker w arrives ("true" judgement in line 3), we make a new plan for w from the $freeTaskSet$, as shown in lines 4-5. Otherwise if a task t appears, we try to combine t with the task sequence (plan) of a worker in $aWorkerSet$, with minimized increased travel distance (lines 8-15). If such combination does not exist, t is added to $freeTaskSet$ and waits to be assigned to prospective workers, as shown in lines 16-17.

Example 4. We use the settings in Example 1 to run the Fast-Planning algorithm. At time 1, w_1 arrives and moves to t_1, which is the same as in the Delay-Planning algorithm. At time 1.5, t_2 appears and we try to combine it with w_1's task sequence S_{w_1}. At this time, w_1 is at $(1.22, 1.45)$. With simple calculation, it results in a smaller increased travel distance by performing t_2 first than performing t_1 first. Therefore $S_{w_1} = \langle t_2, t_1 \rangle$. At time 3, w_2 arrives. Now $freeTaskSet = \{t_3\}$, but w_2 cannot accomplish t_3 on time. Hence w_2 directly moves to his/her destination d_{w_2}. t_4 and t_5 appear at time 5 and 5.5, respectively, and only t_4 can be accomplished by w_2. This process is similar to that in the example of the Delay-Planning algorithm and we omit the details. Finally, t_1, t_2 and t_4 are accomplished, and we obtain a utility value of 9.

Time Complexity. We still use n and m to denote the number of workers and tasks, respectively. Lines 3-5 are executed at most $O(n)$ times, and the time complexity per execution is $O(m \log m)$. Thus the total time complexity of lines 3-5 is $O(mn \log m)$. When the request is a task, lines 11-13 are executed $O(nm^2)$ times ($O(n)$ for line 9, and $O(m)$ for line 6 and line 10). Line 8 and lines 14-17 can be executed in $O(1)$ time, and they are iterated at most $O(m)$ times. The total time complexity of lines 6-17 is $O(nm^2)$. Combing these two parts, the time complexity of the Fast-Planning algorithm is $O(nm^2)$.

4 Experimental Study

4.1 Experimental Setup

Datasets. We evaluate the performance of the proposed algorithms on both synthetic and real datasets. Table 2 shows the settings of the synthetic dataset, where the default settings are marked in bold. Tasks and workers are randomly

Algorithm 3: Fast-Planning

input : A set of workers W, a set of tasks T
output: Plans for $w \in W$

$aWorkerSet \leftarrow \emptyset, freeTaskSet \leftarrow \emptyset$;
for *each new arrival request* **do**
  if *the request is a worker* w **then**
    `BenefitGreedy`$(w, freeTaskSet)$;
    $aWorkerSet \leftarrow aWorkerSet \cup \{w\}$;
  else
    // Denote the arrival task by t.
    $w_{best} \leftarrow None, bestComPos \leftarrow -1, minCost \leftarrow \infty$;
    foreach $w_a \in aWorkerSet$ **do**
      foreach *combination position* $ComPos$ *in* S_{w_a} **do**
        $tmpCost \leftarrow$extra distance if combine t to $ComPos$;
        if $tmpCost < minCost$ **then**
          $minCost \leftarrow tmpCost, bestComPos \leftarrow ComPos, w_{best} \leftarrow w_a$;
    if $minCost < \infty$ **then**
      Combine t with $S_{w_{best}}$ according to $bestComPos$.
    else
      $freeTaskSet \leftarrow freeTaskSet \cup \{t\}$;

Table 2: Experiments Settings

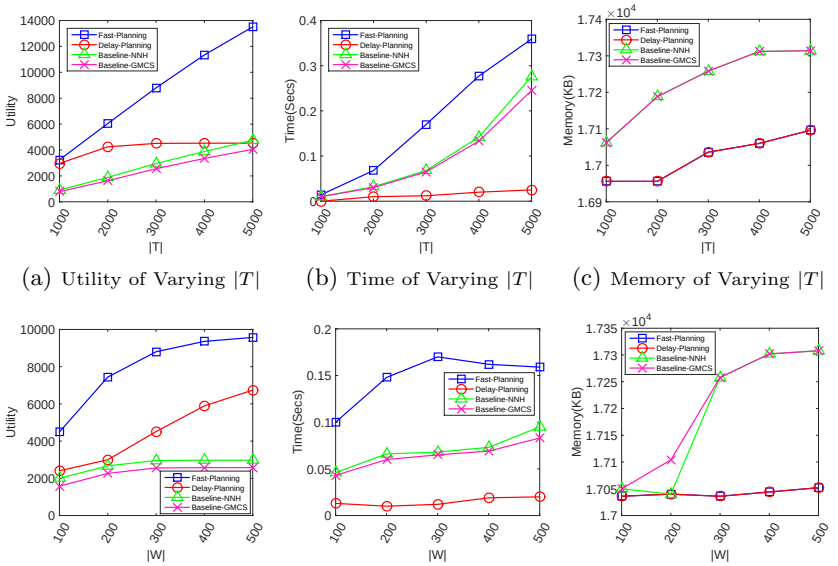
$\|T\|$	$100, 200, \mathbf{300}, 400, 500$
$\|W\|$	$1000, 2000, \mathbf{3000}, 4000, 5000$
ts_t	$\sigma = \mathbf{10}, \mu = 30, 60, \mathbf{90}, 120, 150$
ts_w	$\sigma = \mathbf{10}, \mu = 60, 120, \mathbf{180}, 240, 300$
U_{max}	$2, 4, \mathbf{6}, 8, 10$
Scalability($\|T\| \times \|W\|$)	$10k \times 1k$,$20k \times 2k$,$30k \times 3k$,$40k \times 4k$, $50k \times 5k$,$60k \times 6k$,$70k \times 7k$,$80k \times 8k$, $90k \times 9k$,$100k \times 10k$,$200k \times 20k$,$300k \times 30k$, $400k \times 40k$,$500k \times 50k$

sampled on a 600×600 metric space, with different values of $|T|$ and $|W|$. We also change the extra expiration time span of tasks and workers (ts_t and ts_w). Motivated by [19], the waiting time of a task (worker) follows a Gaussian distribution with the settings as in Table 2. The utilities of tasks are randomly sampled between $[1, U_{max}]$. Similarly to [20], we generate the release time of tasks and workers by the Poisson distribution, with a parameter $\lambda = 2/min$ for workers, and $\lambda = 20/min$ for tasks. We also generate datasets with large scales to test the scalability of the algorithms. For real data, we use the taxi order data, collected from a real taxi-calling service platform, to generate the locations of workers and tasks. Specifically, the location of a task is generated from an order's starting location. The initial location and the destination of a worker are

generated from an order's starting location and destination, respectively. Other settings are the same as in the synthetic dataset.

Baselines. In addition to the two proposed algorithms, we also evaluate the performance of two baseline algorithms. The first is the NNH algorithm in [9], and the second is the GMCS algorithm in [13]. Both of them solve the single-worker task planning problem, and perform best in [9] and [13] respectively. To extend them to the MWATP problem, whenever a task appears, we find a candidate worker set (satisfying the expiration constraint) and randomly assign the task to a worker in the set. Each worker runs the corresponding single-worker algorithm to accomplish the tasks. The two baselines are denoted by Baseline-NNH and Baseline-GMCS, respectively.

Implementation. All the algorithms are implemented in C++, and the experiments were performed on a machine with 40 Intel(R) Xeon(R) E5 2.30GHz CPUs and 512GB memory.



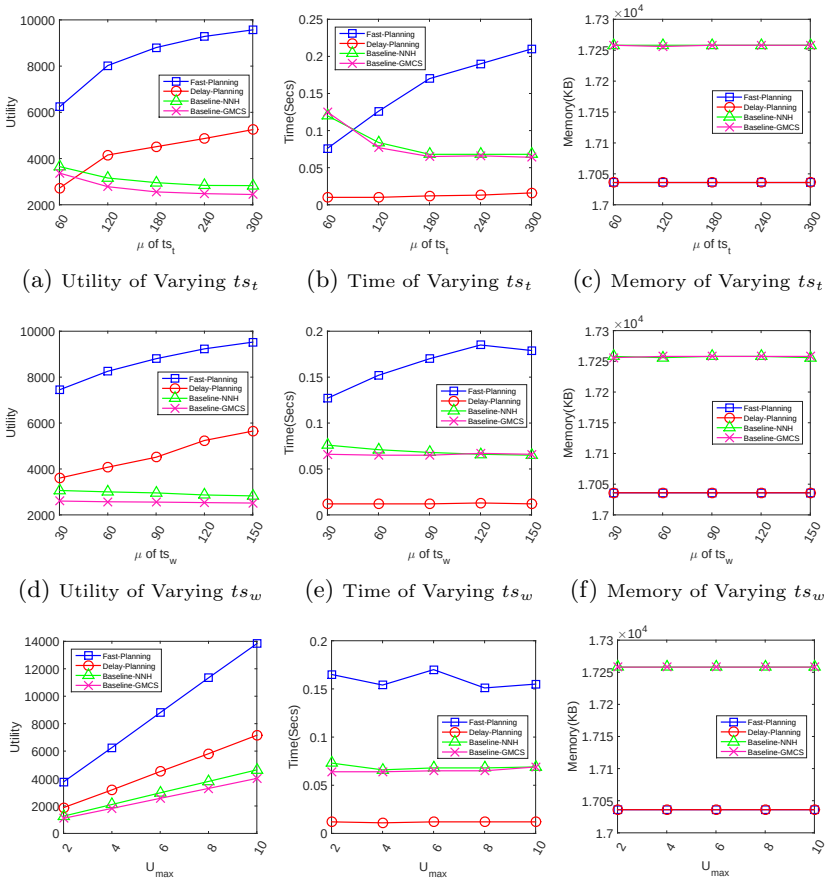
(a) Utility of Varying $|T|$ (b) Time of Varying $|T|$ (c) Memory of Varying $|T|$

(d) Utility of Varying $|W|$ (e) Time of Varying $|W|$ (f) Memory of Varying $|W|$

Fig. 3: Results on Varying $|T|$ and $|W|$

4.2 Experiment Results

Effect of $|T|$. Fig. 3a-Fig. 3c show the results of varying $|T|$. Delay-Planning and Fast-Planning outperform the two baselines in terms of the total utility value while Fast-Planning performs the best. The utility obtained by Delay-Planning is stable, while that of the other three increases with $|T|$. This might be because in each batch of Delay-Planning, tasks have been overflowed, and more tasks do not increase the utility. All the algorithms consume more time when $|T|$ increases, because more tasks lead to a larger searching space. Delay-Planning is the most

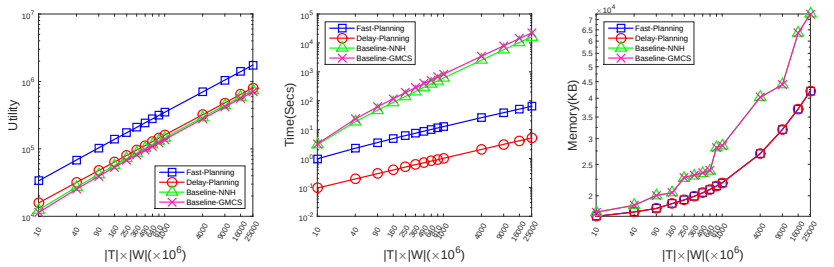


(a) Utility of Varying ts_t (b) Time of Varying ts_t (c) Memory of Varying ts_t

(d) Utility of Varying ts_w (e) Time of Varying ts_w (f) Memory of Varying ts_w

(g) Utility of Varying U_{max} (h) Time of Varying U_{max} (i) Memory of Varying U_{max}

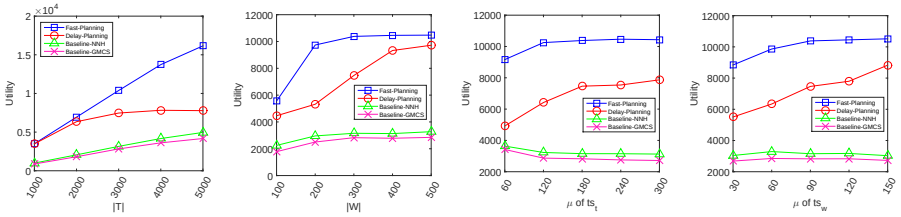
Fig. 4: Results on Varying ts_t, ts_w and U_{max}



(a) Utility of Varying $|T||W|$ (b) Time of Varying $|T||W|$ (c) Memory of Varying $|T||W|$

Fig. 5: Results on Scalability

time-efficient, while Fast-Planning consumes more time, because a combination inspection for all workers is called whenever a task appears. For memory, Delay-Planning and Fast-Planning consume more space when $|T|$ increases, but are still more efficient than baselines.



(a) Varying $|T|$ (b) Varying $|W|$ (c) Varying ts_t (d) Varying ts_w

Fig. 6: Utility Results on Real Datasets

Effect of $|W|$. Fig. 3d-Fig. 3f show the results of varying $|W|$. For the total utility value, Delay-Planning and Fast-Planning perform better than the baselines. The total utility of Delay-Planning and Fast-Planning increases with $|W|$, yet that of the baselines remain almost constant. The running time of all the algorithms are stable as the increase of $|W|$. This might be because the tasks are overflowed, and more tasks do not lead to more efficient plans. Delay-Planning is still the most time-efficient. For memory, Delay-Planning and Fast-Planning consume stable space as $|W|$ increases, which is better than the baselines.

Effect of ts_t. Fig. 4a-Fig. 4c show the results of varying ts_t. The total utility of Delay-Planning and Fast-Planning increases as ts_t increases, but that of the baselines decrease as ts_t increases. This is probably because when μ of ts_t increases, a worker can be assigned to a task far away from him/her, which wastes too much time in one task, which leads to a decrease in the total utility. Delay-Planning and Fast-Planning still get larger total utility than the baselines (except for Delay-Planning when $\mu = 60$), and Fast-Planning performs the best. As μ of ts_t increases, the running time of Delay-Planning and Fast-Planning increases, but that of the baselines tends to be stable. This is because the size of the candidate task set for a worker is restricted by the spare time of workers, even though tasks have more waiting time. All the four algorithms consume stable memory, but the two proposed algorithms require less memory.

Effect of ts_w. Fig. 4d-Fig. 4f show the results of varying ts_w. As μ of ts_w increases, the total utility values of our algorithms increase, because workers have more spare time to accomplish tasks. Delay-Planning and Fast-Planning perform better than the baselines. As μ of ts_w increases, the running time of Delay-Planning and Fast-Planning increases, while that of the baselines tend to be stable. The reason is similar as when varying ts_t. Delay-Planning is still the most time-efficient. The memory of the four algorithms are similar as when varying ts_t.

Effect of U_{max}. Fig. 4g-Fig. 4i show the results of varying U_{max}. The total utility value of all the algorithms increases linearly as U_{max} increases. The running time of all the algorithms remains stable, indicating the utility of tasks has no impact on the running time. The trend of the memory consumption is similar as when ts_t varies.

Scalability. The scalable results are shown in Fig. 5. The utility and the running time of the four algorithms increase linearly as $|T| * |W|$ increases, and

our algorithms perform better than baselines. Our algorithms also consume less memory than the baselines.

Performance on Real Datasets. Fig. 6 shows the results of the total utility value on real datasets. The results are similar to those on the synthetic datasets. The results for the running time and memory are also similar to those on the synthetic datasets. Thus we omit the figures of memory and time due to the limited space.

Summary of Results. The Delay-Planning algorithm, though performs worse than the Fast-Planning algorithm in terms of the total utility value, has the most efficient running time. The Fast-Planning algorithm obtains the largest total utility at the cost of a slightly longer running time than the Delay-Planning algorithm. Both the two proposed algorithms can fit the scalable environment in terms of total utility and running time. The results are also similar on the real-world datasets.

5 Related Work

Our work is related to the domains of **Spatial Crowdsourcing** and **Orienteering Problem**.

5.1 Spatial Crowdsourcing

Spatial crowdsourcing has attracted extensive research interest since [2].

Task Assignment and Planning. Task assignment, task planning in particular, is one of the most important issues in spatial crowdsourcing [2][9] [21][22][23]. In [11], the authors make plans for a single worker in the offline scenario, with the objective to maximize the number of accomplished tasks. The model is generalized to multiple workers in [12][15], but still only for the offline scenario. Both models [11][12] try to find approximate plans for workers. In [22], a protocol is proposed for protecting the privacy while task assignment. One recent work [16] makes one step further to find the exact plans of maximizing accomplished tasks for the offline scenario by using dynamic programming and graph partition.

Online Models. Since many real-world spatial crowdsourcing applications are real-time, recent studies have proposed various online models. In [24] and [25], the authors study the maximizing weighted bipartite matching in the one-sided online scenario, where only nodes on one side appear dynamically. The two-sided online scenario is explored in [10], and a solution with a competitive ratio of $\frac{1}{4}$ is proposed. [23] further studies the online trichromatic matching problem. However, these works focus on task assignment as a bipartite matching problem, which is invalid for task planning in our work. The closest related work is [13], which studies the route planning problem for a single worker in the one-sided online scenario (*i.e.*, only tasks appear dynamically).

5.2 Orienteering Problem

Given a worker with a starting location, an end location, and a time budget, and a set of n nodes in the plane, each of which is associated with a score, the

orienteering problem aims to make a scheduling for the worker to gain maximal scores, with the constraint of costing less time than the time budget [17]. Many variants of the orienteering problem have been proposed [26]. Among them, the Team Orienteering Problem with Time Windows (TOPTW) [27] is the closest to our work. In this problem, each node is associated with a valid time window and we need to find a proper scheduling for a team of workers. However, these time windows can be observed at the beginning of the system, which means that the TOPTW is still an offline scenario. Furthermore, in TOPTW the workers are foreknown, while in our MWATP problem, the arrivals of both workers and tasks are unknown beforehand.

6 Conclusion

In this paper, we propose a new online task planning problem, called *Multi-Worker-Aware Task Planning* (MWATP) problem. We prove that the offline MWATP problem is NP-hard and no online algorithm has a constant competitive ratio. We then propose two heuristic algorithms, called Delay-Planning and Fast-Planning to solve the MWATP problem. We finally evaluate the effectiveness and the efficiency of the proposed algorithms on both synthetic and real datasets.

Acknowledgment

Qian Tao, Yongxin Tong and Ke Xu's works are partially supported by the National Science Foundation of China (NSFC) under Grant No. 61502021 and 71531001, National Grand Fundamental Research 973 Program of China under Grant 2014CB340300, the Base construction and Training Programme Foundation for the Talents of Beijing under Grant No. Z171100003217092, and the Science and Technology Major Project of Beijing under Grant No. Z171100005117001. Yuxiang Zeng and Lei Chen's works are partially supported by the Hong Kong RGC GRF Project 16207617, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Webank Collaboration Research Project, and Microsoft Research Asia Collaborative Research Grant.

References

1. Tong, Y., Chen, L., Zhou, Z., Jagadish, H.V., Shou, L., Lv, W.: Slade: A smart large-scale task decomposer in crowdsourcing. IEEE Transactions on Knowledge and Data Engineering (2018)
2. Kazemi, L., Shahabi, C.: Geocrowd: enabling query answering with spatial crowdsourcing. In: GIS. (2012) 189–198
3. Zeng, Y., Tong, Y., Chen, L., Zhou, Z.: Latency-oriented task completion via spatial crowdsourcing. In: ICDE. (2018)
4. Tong, Y., Chen, Y., Zhou, Z., Chen, L., Wang, J., Yang, Q., Ye, J., Lv, W.: The simpler the better: A unified approach to predicting original taxi demands on large-scale online platforms. In: SIGKDD. (2017) 1653–1662
5. Chen, L., Shahabi, C.: Spatial crowdsourcing: Challenges and opportunities. IEEE Data Engineering Bulletin **39**(4) (2016) 14–25
6. Tong, Y., Chen, L., Shahabi, C.: Spatial crowdsourcing: Challenges, techniques, and applications. PVLDB **10**(12) (2017) 1988–1991

7. Tong, Y., She, J., Ding, B., Chen, L., Wo, T., Xu, K.: Online minimum matching in real-time spatial data: Experiments and analysis. PVLDB **9**(12) (2016) 1053–1064
8. She, J., Tong, Y., Chen, L., Cao, C.C.: Conflict-aware event-participant arrangement and its variant for online setting. IEEE Transactions on Knowledge and Data Engineering **28**(9) (2016) 2281–2295
9. Kazemi, L., Shahabi, C., Chen, L.: Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In: GIS. (2013) 304–313
10. Tong, Y., She, J., Ding, B., Wang, L., Chen, L.: Online mobile micro-task allocation in spatial crowdsourcing. In: ICDE. (2016) 49–60
11. Deng, D., Shahabi, C., Demiryurek, U.: Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In: GIS. (2013) 314–323
12. Deng, D., Shahabi, C., Zhu, L.: Task matching and scheduling for multiple workers in spatial crowdsourcing. In: GIS. (2015) 21:1–21:10
13. Li, Y., Yiu, M.L., Xu, W.: Oriented online route recommendation for spatial crowdsourcing task workers. In: SSTD. (2015) 137–156
14. She, J., Tong, Y., Chen, L.: Utility-aware social event-participant planning. In: SIGMOD. (2015) 1629–1643
15. Deng, D., Shahabi, C., Demiryurek, U., Zhu, L.: Task selection in spatial crowdsourcing from worker's perspective. GeoInformatica **20**(3) (2016) 529–568
16. Zhao, Y., Li, Y., Wang, Y., Su, H., Zheng, K.: Destination-aware task assignment in spatial crowdsourcing. In: CIKM. (2017) 297–306
17. Golden, B.L., Levy, L., Vohra, R.: The orienteering problem. Naval research logistics **34**(3) (1987) 307–318
18. Yao, A.C.C.: Probabilistic computations: Toward a unified measure of complexity. In: FOCS. (1977) 222–227
19. 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) (2017) 1334–1345
20. Roy, S.B., Lykourentzou, I., Thirumuruganathan, S., Amer-Yahia, S., Das, G.: Task assignment optimization in knowledge-intensive crowdsourcing. The VLDB Journal **24**(4) (2015) 467–491
21. To, H., Shahabi, C., Kazemi, L.: A server-assigned spatial crowdsourcing framework. ACM Transactions on Spatial Algorithms and Systems **1**(1) (2015) 2:1–2:28
22. Liu, A., Wang, W., Shang, S., Li, Q., Zhang, X.: Efficient task assignment in spatial crowdsourcing with worker and task privacy protection. Geoinformatica (3) (2017) 1–28
23. Song, T., Tong, Y., Wang, L., She, J., Yao, B., Chen, L., Xu, K.: Trichromatic online matching in real-time spatial crowdsourcing. In: ICDE. (2017) 1009–1020
24. Mehta, A.: Online matching and ad allocation. Foundations and Trends in Theoretical Computer Science **8**(4) (2013) 265–368
25. Ting, H., Xiang, X.: Near optimal algorithms for online maximum edge-weighted b-matching and two-sided vertex-weighted b-matching. Theoretical Computer Science **607** (2015) 247–256
26. Gunawan, A., Lau, H.C., Vansteenwegen, P.: Orienteering problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research **255**(2) (2016) 315–332
27. Vansteenwegen, P., Souffriau, W., Berghe, G.V., Oudheusden, D.V.: Iterated local search for the team orienteering problem with time windows. Computers & OR **36**(12) (2009) 3281–3290