Team-Oriented Task Planning in Spatial Crowdsourcing

Dawei Gao[1], Yongxin Tong[1], Yudian Ji[2], and Ke Xu[1]

[1] SKLSDE Lab and IRC, Beihang University, China
[2] The Hong Kong University of Science and Technology, Hong Kong SAR, China
[1]{david_gao,yxtong,kexu}@buaa.edu.cn, [2]yjiab@connect.ust.hk

Abstract. The rapid development of mobile devices has stimulated the popularity of spatial crowdsourcing. Various spatial crowdsourcing platforms, such as Uber, gMission and Gigwalk, are becoming increasingly important in our daily life. A core functionality of spatial crowdsourcing platforms is to allocate tasks or make plans for workers to efficiently finish the published tasks. However, existing studies usually ignore the fact that tasks may impose different skill requirements on workers, which may lead to decreased numbers of accomplished tasks in real-world applications. In this work, we propose a practical problem called TOTP, *T*eam-*O*riented *T*ask *P*lanning, which not only makes feasible plans for workers but also satisfies the skill requirements of different tasks on workers. We prove the NP-hardness of TOTP, and propose two greedy-based heuristic algorithms to solve the TOTP problem. Evaluations on both synthetic and real-world datasets verify the effectiveness and the efficiency of the proposed algorithms.

Keywords: Spatial crowdsourcing, Task plan, Team formation.

1 Introduction

With the rapid development of mobile and intelligent devices, spatial crowdsourcing platforms, such as Uber, gMission [3] and Gigwalk, are gaining increasing popularity. Different from traditional crowdsourcing platforms, tasks published on spatial crowdsourcing platforms require workers to travel to specific locations to accomplish the tasks.

A fundamental issue in spatial crowdsourcing is the planning problem [12, 14], which refers to making traveling plans for workers to efficiently finish the published tasks under constraints such as travel budgets and completion time. We argue that such a problem formulation is impractical, because the tasks on real-life spatial crowdsourcing platforms often come with various requirements. Consequently, only workers with the desired skills are able to accomplish the corresponding tasks. Imagine the following scenario. There is a spatial crowdsourcing platform which provides domestic services. Currently it has three tasks: the first one needs cleaning and tutoring from 3:00 p.m. to 5:00 p.m.; the second requires babysitting and cleaning from 4:00 p.m. to 6:00 p.m.; and the third needs cooking from 6:00 p.m. to 7:00 p.m. There are also some workers on the

Table 1: Basic Information of Tasks and Workers in Example 1

Workers			Tasks			
No	Owning Skills	Travel budget	No	Required Skills	Capacity	Time period
w_1	$\{e_2, e_4\}$	24	t_1	$\{e_2, e_3\}$	2	[5,6]
w_2	$\{e_3\}$	20	t_2	$\{e_2\}$	2	[1,3]
w_3	$\{e_3, e_4\}$	19	t_3	$\{e_2, e_4\}$	1	[7,8]
w_4	$\{e_1, e_2\}$	21	t_4	$\{e_1, e_2, e_3\}$	2	[2,4]
w_5	$\{e_1, e_4\}$	23				

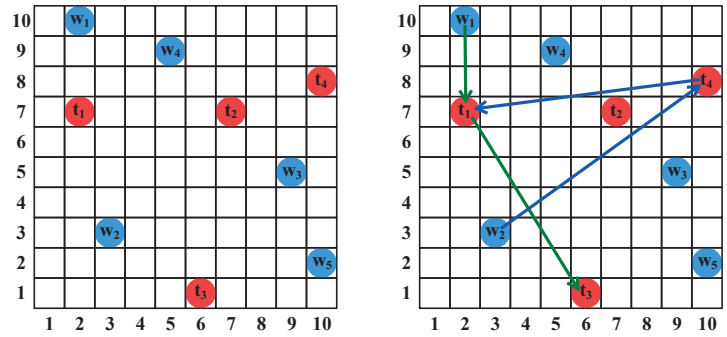
Table 2: Satisfaction between Tasks and Workers in Example 1

	w_1	w_2	w_3	w_4	w_5
t_1	1	3	2	1	5
t_2	2	3	2	1	4
t_3	5	2	3	4	1
t_4	4	4	2	1	6

platform: Paul is skilled at cleaning and babysitting, David is good at cooking and Lucy is skilled at tutoring. Note that it is non-trivial for a single worker, such as Paul or David, to accomplish the requirement of the above tasks. It is also difficult for the platform to make plans for the workers under the spatial and time constraints.

Existing solutions to the planning problems in spatial crowdsourcing do not consider the skill requirements, spatial and time constrains simultaneously. In the above scenario, existing studies will assign the first task to Paul, which will not be completed. To jointly account for the skill requirements of tasks and the spatial and time constraints, our key insight it to assign *a team of workers* to fulfil all the requirements. We propose TOTP, a Team-Oriented Task Planning problem to maximum the total satisfaction of the workers. Note that we use skills to represent the specific requirements of tasks on workers. We illustrate the motivation of TOTP with the following example. In this example, the skills are denoted as $\{e_1, \cdots, e_4\}$.

Example 1. Suppose we have five workers $w_1 - w_5$ and four tasks $t_1 - t_4$ on a spatial crowdsourcing platform. The locations of the workers and the tasks are shown in the 2D space in Fig. 1a. We use Euclidean distance in this example. Table 1 shows the attributes of the workers and the tasks. The skills of the workers and the distances that he/she would like to travel are shown in the second and third columns. The skill requirements of tasks on workers are shown in the fifth column. Capacity shows the maximum number of workers that can participate in the corresponding task. The last column in Table 1 shows the completion time, which is the duration that the assigned worker needs to stay at the task's location. Table 2 shows the satisfaction of workers, which represents the workers' preferences on the tasks. The spatial crowdsourcing platform has to make assignments between the workers and tasks such that the



(a) Initial Locations of Tasks/Workers (b) A Global Task Planning

Fig. 1: Example 1.

skill requirements of tasks are satisfied and the total satisfaction is maximized. Fig. 1b shows a global task planing, i.e. $\{t_1, t_3\}$ for w_1 and $\{t_4, t_1\}$ for w_2, respectively. Notice that t_1 can be accomplished by a team of w_1 and w_2.

Contributions. We propose a more realistic planning problem in spatial crowdsourcing called the *Team*-*Oriented* *Task* *Planning* problem (TOTP). As the example indicates, the TOTP problem not only makes plans for each worker but also attempts to satisfy the skill requirements of different tasks. To summarize, our contributions are as follows.

- We identify TOTP, a new spatial crowdsourcing planning problem that accounts for the skill requirements of tasks on workers.
- We prove that the TOTP problem is NP-hard.
- We propose two greedy algorithms to solve the TOTP problem, and analyze the complexity of both algorithms.
- We verify the effectiveness and efficiency of the proposed algortihms through extensive experiments on synthetic and real-world datasets.

In the rest of the paper, we review related work in Section 2, formulate the TOTP problem and prove its NP-hardness in Section 3. Section 4 presents our algorithms on TOTP problem and Section 5 show the experimental evaluations. Finally we conclude this paper in Section 6.

2 Related Work

Our TOTP problem is closely related to two categories of research: *spatial crowdsourcing* and *team formation*.

2.1 Spatial Crowdsourcing

The task assignment problem is fundamental in spatial crowdsourcing. Many efforts aim to maximize the total number or total utility of tasks that are assigned to workers in static scenarios [9, 19]. Others study the

conflict-aware spatial task assignment problems [14, 15, 22]. Recently, the problem of online task assignment in dynamic spatial crowdsourcing was first proposed by [21], and several variants of online task assignment in dynamic scenarios were also studied in [16, 17, 20]. Other practical issues such as location privacy protection of workers have also been explored [18]. In addition, [6] introduced the route planning problem for a worker and attempt to maximize the number of complete tasks, while the corresponding online version of [6] is investigated in [11]. [7] studies to assign workers under the spatial and time constraints. [3] summarizes the challenges and opportunities in spatial crowdsourcing. Despite the extensive research efforts on task assignment in spatial crowdsourcing, they all assume simple and homogenous tasks without considering the situations where the tasks are complex and require a team of workers to finish.

2.2 Team Formation

The team formation problem is first proposed by Lappas in [10], which aims to find the minimum cost team of experts according to the skills and relationships of users in social networks. Notice that the team formation problem can be reduced from typical NP-complete problems, indicating that the team formation problem is NP-hard. [1] focuses on minimizing the maximum workload when forming teams to cover the skills, and studies both the off-line and on-line settings. [13] studies the team formation problem with capacity constraints on a social network, and presents approximation algorithms with provable guarantees. [2] studies the online team formation problem called the Balanced Social Task Assignment problem, and proposes an online algorithm with provable guarantee. In [8], based the skills of crowd workers, the authors study how to recommend k teams for spatial crowdsourcing tasks. Furthermore, Cheng et al. also proposed the issue of team-oriented task assignment in spatial crowdsourcing recently [5]. The above studies only focus on satisfying the skill requirements. They neither consider the location information and travel budgets, nor address the problem of how to make feasible plans for workers.

3 Problem Statement

In this section we formally define the Team-Oriented Task Planning (TOTP) problem and prove that the problem is NP-hard. We assume $E =< e_1, \cdots, e_m >$ as a universe of m skills throughout this paper. Let T be a set of tasks. Each task t has its own location l_t that requires the assigned workers to travel to. The task t's skill requirement is represented by a subset of E: $E_t = \{e_1, \cdots, e_{|E_t|}\}$. We also define a time interval $[s_1^t, s_2^t]$, which is the starting time and the ending time of the corresponding task t. Note that the assigned workers need to stay at l_t during this time interval. In real-life applications, the number of workers to finish a task is normally limited. Hence we define the task's capacity c_t as the maximum number of workers allowed for the task. We define W

as a set of workers. Each worker w has his/her starting location l_w, a set of skills $E_w = \{e_1, \cdots, e_{|E_w|}\}$ and a travel budget B_w, which represents the total travel cost that the worker w would like to spend to accomplish the tasks, which can be money, distance or time.

Definition 1 (Crowd Worker). *A crowd worker ("worker" for short) is denoted as $w =< l_w, E_w, B_w >$, where l_w is the starting location of worker w in the 2D space, E_w is a set of skills that the worker is good at, and B_w is the travel budget that the worker w would like to spend to accomplish the tasks.*

Definition 2 (Crowdsourced Task). *A crowdsourced task ("task" for short) is denoted as $t =< l_t, E_t, Int, c_t >$, where l_t is the location in the 2D space that workers have to travel to, E_t is the task's required skills, $Int = [s_1^t, s_2^t]$ represents the task's time interval during which the task should be accomplished and c_t is the capacity of the task that limits the number of the workers assigned to the task.*

For each worker w, we define $P_w = \{t_1^w, t_2^w, \cdots, t_{|P_w|}^w\}$ as the plan of the arranged tasks in time order. Suppose that t_i and t_{i+1} are two tasks in P_w, a plan is feasible if there is no time conflict among the arranged tasks, and the workers can perform t_{i+1} in time after finishing t_i. To evaluate the travel cost between any two tasks, such as t_i and t_j, we use $cost(l_{t_i}, l_{t_j})$ to represent the travel cost between t_i and t_j. If a worker cannot perform the next task in time, the cost between these two tasks will be ∞. Meanwhile, we define $u(w, t)$ as the satisfaction between task t and worker w, and $\mathcal{U}(w) = \sum_{t_i \in P_w} u(w, t_i)$ as worker w's satisfaction on P_w. Finally we give the definition of feasible plan and the Team-Oriented Task Planning (TOTP) problem.

Definition 3 (Feasible Plan). *A plan P_w is feasible if and only if: $s_2^{t_i} \leq s_1^{t_{i+1}}, \forall 1 \leq i \leq |P_w| - 1$.*

Definition 4 (Team-Oriented Task Planning (TOTP) Problem). *Given a set of tasks $T = \{t_1, t_2, \cdots, t_{|T|}\}$ and a set of workers $W = \{w_1, w_2, \cdots, w_{|W|}\}$ with their associated attributes, the Team-Oriented Task Planning (TOTP) problem is to find feasible plans $A = \cup_w \{P_w\}$ for workers with the maximum utility cost: $Utility(A) = \sum_{w_i \in W} \mathcal{U}(w_i)$, such that the following constraints are satisfied:*

- *Skill constraint: each required skill of the tasks is covered by the assigned workers.*
- *Travel budget constraint: a worker's total travel cost is under his/her travel budget.*
- *Capacity constraint: the number of workers assigned to a task is lower than the task's capacity.*

Theorem 1. *The Team-Oriented Task Planning problem is NP-hard.*

Proof. We prove that the Team-Oriented Task Planning problem is NP-hard by reducing the knapsack problem, a well known NP-complete problem, to the TOTP problem. An instance of the knapsack problem consists

Table 3: Summary of Symbol Notations

Notation	Description
w	Worker
t	Task
c_t	Capacity of t
$l_w(l_t)$	Location of w(or t)
T	Set of tasks
W	Set of workers
B_w	Budget of w
$E_w(E_t)$	Skill set of w(or t)
P_w	Plan of w
$A = \cup_w\{S_w\}$	The total plan
$u(t, w)$	satification between t and w
$\mathcal{U}(w) = \sum_{t \in P_w} u(w, t)$	w's satification on P_w
$Utility(A) = \sum_{w \in W} \mathcal{U}(w)$	The total satification of A

of a set of n items $\{x_1, \cdots, x_n\}$ where each item x_i has its value $v_i > 0$, weight $m_i > 0$, and the maximum weight M that the bag can carry. The decision version of the knapsack problem asks whether there is a collection of items $C = \{x_{s_1}, x_{s_2}, \cdots, x_{s_k}\}$ such that $\sum_{i=1}^{k} v_{s_i} = K$ and $\sum_{i=1}^{k} m_{s_i} \leq M$. We construct an instance of the knapsack problem using an instance of the TOTP problem as follows.

- Let $|W| = 1, W = \{w\}$, and $B_w = M$.
- Let $E_t = E_w, \forall w \in W$ and $\forall t \in T$
- Each item corresponds to a task in TOTP problem. $u(w, t_i) = \frac{v_i}{\max v_i}, \forall 1 \leq i \leq n$ and the capacities of all the tasks equal to 1.
- Let $s_2^{t_i} < s_1^{t_{i+1}}, \forall 1 \leq i < n$
- The travel cost of worker w and task t_i is set as: $cost(w, t_i) = \frac{m_i}{2}$
- The travel cost between two events is constructed as:

$$cost(l_{t_i}, l_{t_j}) = \begin{cases} \frac{m_i + m_j}{2} & 1 \leq i < j \leq n \\ +\infty & otherwise \end{cases}$$

Thus, the problem is to decide if there is a feasible plan P_w for w such that $\sum_{t_i \in P_w} u(w, t_i) = \frac{K}{\max v_i}$ satisfying all the constraints. We can see that if the collection exists, then the plan P_w is feasible, and it satisfies that $\sum_{t_i \in P_w} u(w, t_i) = \frac{K}{\max v_i}$ and the total travel cost is less than M. That is, if the plan P_w exists, then there is a collection C satisfying the constraints on the sum of values and weights. □

4 Algorithms of TOTP Problem

In this section, we present two greedy-based algorithms to solve the TOTP problem.

4.1 Rarest Skill Priority Algorithm

We first present a greedy algorithm called the Rarest Skill Priority Algorithm. The Rarest Skill Priority Algorithm recursively selects such skills that are required by numbers of tasks but only few workers have such skills. We call such skills *rarest skill*. The reason why we choose rarest skills in priority is to avoid the case where the workers possessing the rare skills have been assigned to other tasks and the tasks requiring these skills would never be accomplished. The rarest skills are calculated by $\arg\max_{e \in E} \frac{|\{t|e \in E_t\}|}{|\{w|e \in E_w\}|}$. Then, if a number of workers or tasks own/require the rarest skill, we greedily make an assignment of a pair of $(worker, task)$ such that the utility gain is the largest. The utility gain is defined in Equations 1.

$$ratio(w,t) = \frac{u(w,t)}{inc\_cost(w,t)} \tag{1}$$

where $inc\_cost(w,t) =$

$$\begin{cases} cost(l_w, l_t) & P_w = \emptyset \\ cost(l_w, l_t) + cost(l_t, l_{t_1^w}) & \\ \quad -cost(l_w, l_{t_1^w}) & s_2^t < s_1^{t_1^w} \\ cost(l_{t_i^w}, l_t) + cost(l_t, l_{t_{i+1}^w}) & \\ \quad -cost(l_{t_i^w}, l_{t_{i+1}^w}) & s_2^{t_i^w} < s_1^t, s_2^t < s_1^{t_{i+1}} \\ cost(l_{t_{|P_w|}^w}, l_t) & s_2^{t_{|P_w|}^w} < s_1^t \\ \infty & otherwise \end{cases} \tag{2}$$

Equation 1 defines the ratio between the satisfaction and the additional travel cost. With a larger $ratio(w,t)$, the task t is more suitable for w since he/she has a larger satisfaction and less travel cost. In Equation 2, $inc\_cost(w,t)$ is the additional travel distance when inserting task t to plan $P_w = \{t_1^w, t_2^w, \cdots, t_{|P_w|}^w\}$. The details of $inc\_cost(w,t)$ are shown in Equation 2. If P_w is an empty set, the worker only needs to travel to l_t to perform the task. Otherwise, when the starting time of t is earlier than that of the first task of P_w, we can insert t into P_w as the first task. Hence the additional travel cost is $cost(l_w, l_t) + cost(l_t, l_{t_1}^w) - cost(l_w, l_{t_1}^w)$. When t's time interval is between task t_i^w and t_{i+1}^w in P_w ($\forall 1 \leq i \leq |P_w| - 1$), we insert the task t in-between and the worker needs to travel to l_t after finishing task t_i^w. Thus the additional cost is $cost(l_{t_i^w}, l_t) + cost(l_t, l_{t_{i+1}^w}) - cost(l_{t_i^w}, l_{t_{i+1}^w})$. Finally, if task t's starting time is after the last task's ending time, we define the additional cost as $cost(l_{t_{|P_w|}^w}, l_t)$.

Algorithm 1 shows the pseudo-code of the Rarest Skill Priority Algorithm. Specifically, when making plans for the workers possessing the rarest skills, we use a heap H to store the worker-task pair (w,t) with the largest ratio for each task in a decreasing order. Then we can pop the

Algorithm 1: Rarest Skill Priority Algorithm

input : A set of workers W, a set of tasks T and their associated attributes
output: $A = \cup_w \{P_w\}$

1: $H \leftarrow \emptyset$
2: **for** $e_{rare} = \arg\max_{e \in E} \frac{|\{t|e \in E_t\}|}{|\{w|e \in E_w\}|}$ **do**
3: $W_{rare} = \{w|e_{rare} \in E_w\}$
4: $T_{rare} = \{t|e_{rare} \in E_t\}$
5: **for** $t_i \in T_{rare}$ **do**
6: $w = \arg\max_{w \in W_{rare}} ratio(w, t_i)$
7: $H \leftarrow (w, t_i)$
8: **end for**
9: **while** $H \neq \emptyset$ **do**
10: Pop (w, t) from H with the largest ratio
11: Add t to P_w if $\{t\} \bigcup P_w$ is feasible and t is not full of capacity
12: $E_t = E_t - E_w$
13: Update (w, t) in H for each t
14: **end while**
15: **end for**

pair on the top of H and add it into the worker's plan. In Algorithm 1, we first initialize the heap H (Line 1). Then we find the rarest skill, and the set of workers W_{rare} and tasks T_{rare} that require/own the rarest skill (Lines 2-4). We traverse the set T_{rare} and add the pair with the largest ratio (calculated by Equation 1) into the heap H for each task (Lines 5-8), pop the pair (w, t) with the largest ratio in H, and add task t to w's plan if it is feasible (Line 10). Afterwards, we update the skill requirement of t to compute the next rarest skill. Because the additional travel cost of the pair associated with w has changed, we need to update the pairs in H (Line 13) for the next iteration. We pop the top pair with the largest ratio until H is empty, and we continue the loop for finding the rarest skill and making assignments (Lines 2-15).

Example 2. Back to Example 1, we first traverse the tasks $t_1 - t_4$ and workers $w_1 - w_5$, and find that e_2 is the rarest skill. Then we make assignment between $W_{rare} = \{w_1, w_4\}$ and $T_{rare} = \{t_1, t_2, t_3, t_4\}$. For each task t in T_{rare}, we find the worker who has the largest ratio with him/her in W_{rare} and push the worker into the heap. In the first iteration, the ratios are shown in Table 4. We choose the pair with the largest ratio for each task and build the heap $H = \{(w_1, t_1), (w_4, t_2), (w_1, t_3), (w_1, t_4)\}$. We pop the largest pair (w_1, t_3) with the ratio 0.51 and add t_3 to w_1's plan. Then we update the pair for t_1, t_2, t_4 in H and finally in this iteration we get $\{(w_1,t_3),(w_1,t_1),(w_4,t_2),(w_4,t_4)\}$. In the following iterations we repeat the process and finally we get $\{t_1, t_3, t_4\}$ for w_1, $\{t_1, t_2\}$ for w_4 and $\{t_4\}$ for w_5.

Complexity Analysis. In the worst case, Algorithm 1 has to traverse all the skills to cover the requirement. During each iteration, we traverse the rest of skills to find the rarest skill whose time cost is

Table 4: Ratios in the First Iteration

	t_1	t_2	t_3	t_4
w_1	0.33	0.34	0.51	0.49
w_4	0.28	0.35	0.50	0.20

$O(|E|(|W| + |T|))$. Then for each task in W_{rare} we spend $O(|W||T|)$ to find the $(worker, task)$ pair with the largest ratio. Searching through and updating the heap take about $O(|W||T|)$. Thus the time complexity of Algorithm 1 is $O(|E|^2(|W|+|T|)+|E||W||T|)$ in the worst case, where $|E|$ is the number of all skills. The memory cost of Algorithm 1 is mainly to store the heap and the plans for workers, which is $O(|W||T|)$.

4.2 Skill Cover and Utility Priority Algorithm

In this subsection, we present another solution to the TOTP problem, which is called the Skill Cover and Utility Priority Algorithm. In the Skill Cover and Utility Priority Algorithm, we first attempt to cover all the skills of a task like the team formation problem. Specifically, for each task we attempt to form a team to satisfy the skill requirement with a minimal team size. If the team size is smaller than the capacity, in the second step we greedily assign the worker with the largest satisfaction to the tasks. Finally we can obtain a team for the task and satisfy the skill requirements of the tasks.
Algorithm 2 presents the details of the Skill Cover and Utility Priority Algorithm. In lines 1-3, we first sort tasks in an increasing order of staring time and initialize $time_w$ for each worker, which records the available time of the worker. Then we traverse the set of tasks. In line 6 we use function $Dis(.)$ to compute the total travel cost of the plan and pick up the set of workers W' who can participate in task t. In lines 7-11 we initialize the team g and choose the worker who has the most required skills and add him/her to team g. Then if the size of g is smaller than c_t, we add workers with the largest satisfaction to the team until the task's capacity is fully occupied. Finally, we update the workers' plans, budgets, current locations and $time_w$ in lines 16-21 and continue the iteration for the next task. Specifically, when the size of team g is larger than c_t, we abandon the task and go on to the next task.

Example 3. Back to our Example 1. In the first step the tasks are sorted as $\{t_2, t_4, t_1, t_3\}$. For task t_2, w_3 has all the required skills. We add w_3 to team g and at this time the team size is smaller than 2. Thus, w_5 is added to the team for having the largest satisfaction for t_2. Then for t_4, we find worker w_4 who has the most required skills and add w_4 into the team. Because w_3 has participated in t_2, which conflicts with t_4, we choose w_2 to cover the last skills. At last the team size equals to t_4's capacity. For t_1 and t_3, we run Algorithm 2 in a similar way, and finally we get a team $\{w_1, w_2\}$ for t_1 and $\{w_1\}$ for t_3. The final result is presented in Table 5 and the total satisfaction is 20.

Algorithm 2: Skill Cover and Utility Priority Algorithm

input : A set of workers W, a set of tasks T and their associated attributes
output: $A = \cup_w \{P_w\}$

1: $sort(T)$
2: **for** w in W **do**
3: $time_w = 0$
4: **end for**
5: **for** t in T **do**
6: $W' = \{w | w \in W$ and $Dis(P_w \cup \{t\}) \leq B_w$ and $time_w \leq s_1^t\}$
7: $g = \emptyset$
8: **while** g cannot cover E_t **do**
9: $w = \text{argmax}_{w \in W'} \{|E_t \cap E_w|\}$
10: $g = g + \{w\}$
11: **end while**
12: **while** $|g| < c_t$ **do**
13: $w = \arg\max_{w \in W' - g} u(w, t)$
14: $g = g + \{w\}$
15: **end while**
16: **for** w in g **do**
17: Add t to P_w
18: $B_w = B_w - cost(w, t)$
19: $l_w = l_t$
20: $time_w = s_2^t$
21: **end for**
22: **end for**
23: **return** $A = \cup_w \{P_w\}$

Table 5: Result of Example 1

worker	w_1	w_2	w_3	w_4	w_5
plan	$\{t_1, t_3\}$	$\{t_4, t_1\}$	$\{t_2\}$	$\{t_4\}$	$\{t_2\}$

Complexity Analysis. For the two-step greedy algorithm, the time cost to sort tasks and initialize the worker set is $O(|T|ln(|T|) + |W|)$. Then we traverse each task to find a feasible team. Forming team g in lines 8-15 takes $O(|E||W|)$ in the worst case. Finally updating the workers' attributes takes $O(|W|)$. Therefore the overall time cost of the algorithm is $O(|T||E||W| + |T||W|)$. The major space cost of Algorithm 2 is from storing the set W', T and their associated attributes, whose overall space cost is $O(|W| + |T|)$.

5 Evaluation

In this section we conduct experiments on both synthetic and real-world datasets. We use the dataset from gMission [4], a spatial crowdsourcing platform, as the real-world dataset. In this dataset, we extract 1000 tasks,

Table 6: Synthetic Datasets

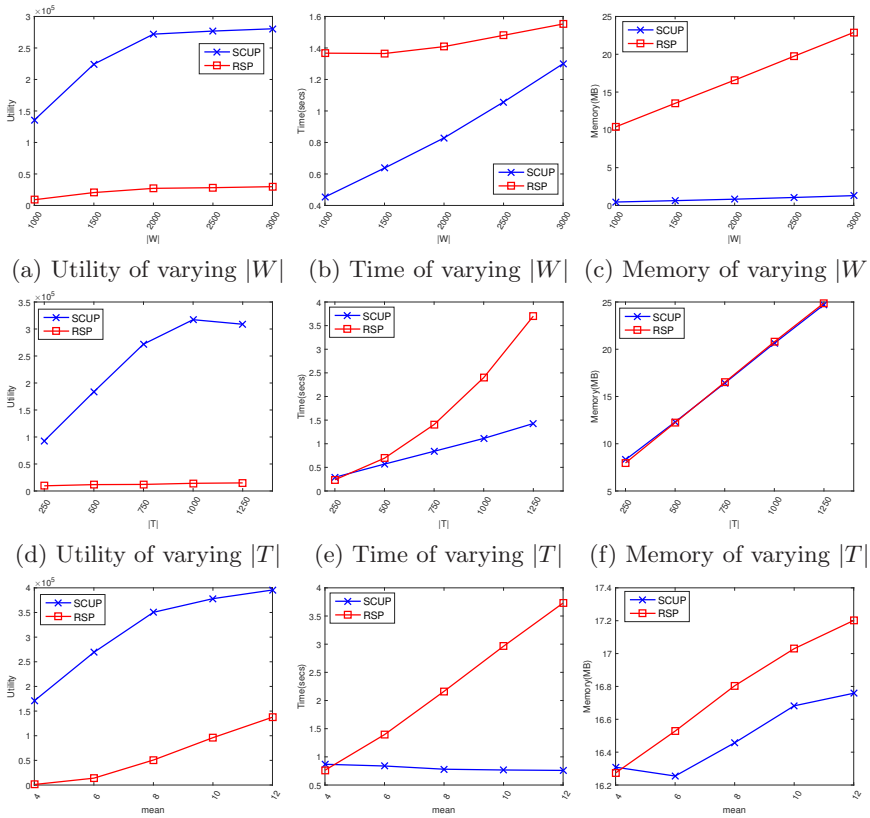
Notation	Value
$\|W\|$	1000,1500,**2000**,2500,3000
$\|T\|$	250,500,**750** ,1000,1250
$mean$	4,6,**8**,10,12
$factor$	0.5,1,**2**,4,8
$\|E_w\|$	3,**6**,9,12,15
$\|E_t\|$	5,10,**15**,20,25

each of which is associated with some descriptions introducing the details. Thus, the required skills of the tasks can be extracted from the descriptions, and the worker's skills are learned from his/her history tasks. Table 6 shows the parameters of the synthetic dataset, and the default values are shown in bold. In the synthetic dataset, the numbers of workers $|W|$ and tasks $|T|$ are set between 1000-3000 and 250-1250, respectively. According to the real-world dataset, we let the capacities of workers follow the normal distribution, with the mean between 4-12. In terms of travel budget B_w, we define a parameter $factor$ to vary the travel budget. Then we have $B_w = \min_{t \in T} cost(l_w, l_t) + \frac{\min_{t \in T} cost(l_w, l_t) + \max_{t \in T} cost(l_w, l_t)}{2} * factor$, and we vary $factor$ from 0.5 to 8. Note that the moving distance of a worker is computed in Euclidean distance and it can be easily extended to the road network distance or other distance metrics. Both the numbers of workers' skills $|E_w|$ and tasks' required skills $|E_t|$ follow the normal distribution and the means are between 3-15 and 5-20. In the real-world dataset, we still use $B_w = \min_{t \in T} cost(l_w, l_t) + \frac{\min_{t \in T} cost(l_w, l_t) + \max_{t \in T} cost(l_w, l_t)}{2} * factor$ as the budget of workers, because there are no such parameters in the real-world dataset.

We evaluate both the Skill Cover and Utility Priority Algorithm (Algorithm 2), denoted as SCUP and the Rarest Skill Priority Algorithm (Algorithm 1), denoted as RSP. We compare these algorithms in terms of utility, time and memory. When comparing utility, we only compute the satisfaction of the completed tasks, whose skills are completely satisfied.

Effect of $|W|$. Fig. 2a to Fig. 2c present the results of varying $|W|$ in the synthetic dataset. The total utility obtained from SCUP is much larger than that from RSP. The running time and the memory cost of both SCUP and RSP are small but increase with $|W|$. The memory of SCUP is smaller than that of RSP, because RSP needs to store the worker-task pairs in the heap.

Effect of $|T|$. Fig. 2d to Fig. 2f show the results of varying $|T|$ in the synthetic dataset. The utility of SCUP increases with $|T|$, and stables when $|T|$ reaches 1000. This is because the number of workers and the travel budget B_w are limited, so the workers cannot complete more tasks. The time cost of SCUP is smaller than that of RSP, and the memory cost of SCUP is approximately equal to that of RSP. Tt is because SCUP spends more time to update the heap and traverse the tasks and workers to find the rarest skill.



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

(d) Utility of varying $|T|$ (e) Time of varying $|T|$ (f) Memory of varying $|T|$

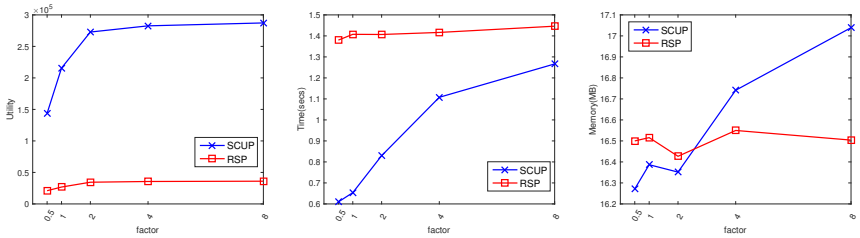
(g) Utility of varying *mean* (h) Time of varying *mean* (i) Memory of varying *mean*

Fig. 2: Results on varying $|W|$, $|T|$, and *mean*.

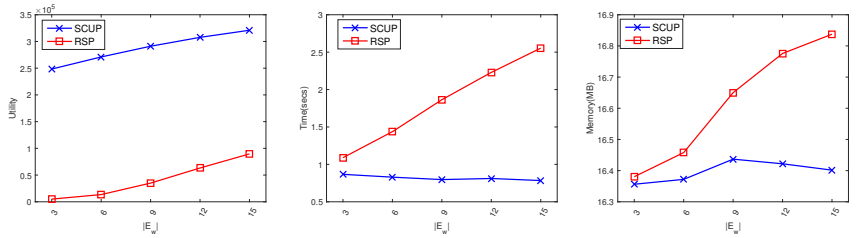
Effect of *mean*. Fig. 2g to Fig. 2i depict the results of varying *mean* in the synthetic dataset. SCUP performs better than RSP not only in total utility but also in time and memory cost. This is because increasing *mean* of capacity enables the tasks to accept more workers. Later the number of workers and the workers' travel budget become the bottlenecks for total utility in SCUP. As for the utility of RSP, the increasing *mean* of capacities increases the possibility of tasks to be accomplished.

Effect of $factor$. Fig. 3a to Fig. 3c present the results of varying $factor$ in the synthetic dataset. The influence of $factor$ on SCUP is stronger than that of RSP (see Fig. 3c). It might be because RSP attempts to cover the rarest skills first, which disperses the workers to different tasks. Thus the utility of RSP increases slowly at the beginning, and increases much faster when the travel budget of workers becomes abundant.

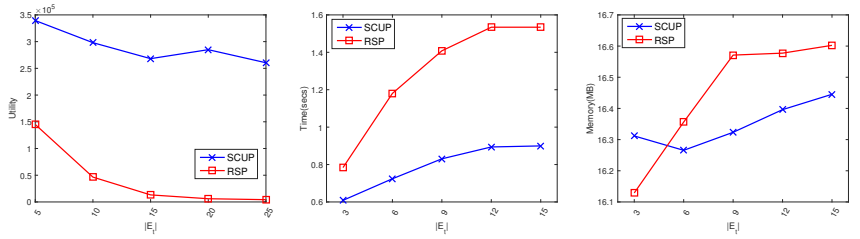
Effect of $|E_w|$. Fig. 3d to Fig. 3f show the results of varying $|E_w|$ in the synthetic dataset. When the workers possess more skills, the possibility of choosing workers with satisfaction for tasks increases. SCUP spends less time to cover the required skills, which results in decreased time cost.



(a) Utility of varying factor (b) Time of varying factor (c) Memory of varying factor



(d) Utility of varying $|E_w|$ (e) Time of varying $|E_w|$ (f) Memory of varying $|E_w|$



(g) Utility of varying $|E_t|$ (h) Time of varying $|E_t|$ (i) Memory of varying $|E_t|$

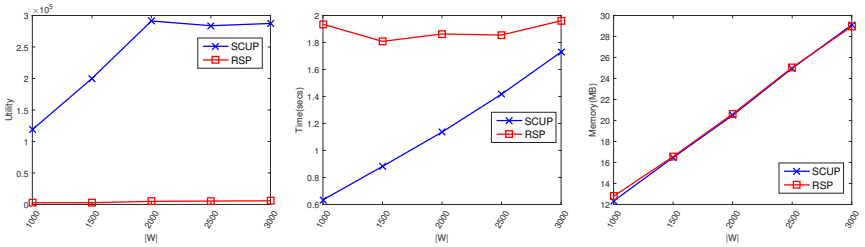
Fig. 3: Results on varying $factor$, $|E_w|$, and $|E_t|$.

Conversely, RSP has to traverse more skills for each worker, which leads to higher time cost.

Effect of $|E_t|$. Fig. 3g to Fig. 3i demonstrate the results of varying $|E_t|$ in the synthetic dataset. When $|E_t|$ increases, the utility of both RSP and SCUP decreases because more workers are needed to satisfy the skill requirements. Due to the extra effort for searching workers to cover the skills, the time and the memory cost also increase with $|E_t|$.

Effect of $|W|$ in real dataset. Finally Fig. 4a to Fig. 4c show the results of varying $|W|$ in the real-world dataset. When $|W|$ increases, the utility of SCUP increases fast at the beginning. When $|W|$ reaches 2000, the utility stabilizes, since the tasks are consumed. For RSP, the utility exhibits a similar but less dynamic trend to SCUP. The memory costs of the two algorithms are approximately the same. The time costs of both algorithms are small.

Summary. SCUP outperforms RSP in utility in various scenarios. One reason is that when making an assignment between workers and tasks owning/requiring the rarest skill, the tasks' capacities are consumed but only few skills are satisfied. Therefore some tasks' skill requirements cannot be completely satisfied. In contrast, SCUP attempts to cover the



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

Fig. 4: Results on real datasets.

skills with a small team of workers, which ensures that the task are actually accomplished. Furthermore, SCUP also outperforms RSP in time and memory.

6 Conclusion

In this paper, we introduce Team-Oriented Task Planning (TOTP) problem, a realistic planning problem in spatial crowdsourcing which attempts to assign workers to suitable tasks. Different from previous research, it also takes into account the skill requirements of tasks on workers. We prove that TOTP is NP-hard, and propose two heuristic algorithms to solve the TOTP problem. Finally we conduct experiments on both synthetic and real-world datasets and verify the effectiveness and the efficiency of the proposed algorithms.

Acknowledgment

This work is supported in part by the National Science Foundation of China (NSFC) under Grant No. 61502021, 61328202, and 61532004, National Grand Fundamental Research 973 Program of China under Grant 2012CB316200.

References

1. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Power in unity: forming teams in large-scale community systems. In: CIKM. pp. 599–608 (2010)
2. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Online team formation in social networks. In: WWW. pp. 839–848 (2012)
3. Chen, L., Shahabi, C.: Spatial crowdsourcing: challenges and opportunities. IEEE Data Engineering Bulletin 39(4), 14–25 (2016)
4. Chen, Z., Fu, R., Zhao, Z., Liu, Z., Xia, L., Chen, L., Cheng, P., Cao, C.C., Tong, Y., Zhang, C.J.: gmission: a general spatial crowdsourcing platform. Proceedings of the VLDB Endowment 7(13), 1629–1632 (2014)

5. Cheng, P., Lian, X., Chen, L., Han, J., Zhao, J.: Task assignment on multi-skill oriented spatial crowdsourcing. IEEE Transactions on Knowledge and Data Engineering 28(8), 2201–2215 (2016)
6. Deng, D., Shahabi, C., Demiryurek, U.: Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In: GIS. pp. 314–323 (2013)
7. Deng, D., Shahabi, C., Zhu, L.: Task matching and scheduling for multiple workers in spatial crowdsourcing. In: GIS. pp. 21:1–21:10 (2015)
8. Gao, D., Tong, Y., She, J., Song, T., Chen, L., Xu, K.: Top-k team recommendation in spatial crowdsourcing. In: WAIM. pp. 191–204 (2016)
9. Kazemi, L., Shahabi, C.: Geocrowd: enabling query answering with spatial crowdsourcing. In: GIS. pp. 189–198 (2012)
10. Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: SIGKDD. pp. 467–476 (2009)
11. Li, Y., Yiu, M., Xu, W.: Oriented online route recommendation for spatial crowdsourcing task workers. In: SSTD. pp. 861–870 (2015)
12. Lu, E.H., Chen, C., Tseng, V.S.: Personalized trip recommendation with multiple constraints by mining user check-in behaviors. In: GIS. pp. 209–218 (2012)
13. Majumder, A., Datta, S., Naidu, K.: Capacitated team formation problem on social networks. In: SIGKDD. pp. 1005–1013 (2012)
14. She, J., Tong, Y., Chen, L.: Utility-aware social event-participant planning. In: SIGMOD. pp. 1629–1643 (2015)
15. She, J., Tong, Y., Chen, L., Cao, C.C.: Conflict-aware event-participant arrangement. In: ICDE. pp. 735–746 (2015)
16. 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), 2281–2295 (2016)
17. Song, T., Tong, Y., Wang, L., She, J., Yao, B., Chen, L., Xu, K.: Trichromatic online matching in real-time spatial crowdsourcing. In: ICDE. pp. 1009–1020 (2017)
18. To, H., Ghinita, G., Shahabi, C.: A framework for protecting worker location privacy in spatial crowdsourcing. Proceedings of the VLDB Endowment 7(10), 919–930 (2014)
19. To, H., Shahabi, C., Kazemi, L.: A server-assigned spatial crowdsourcing framework. ACM Transactions on Spatial Algorithms and Systems 1(1), 2 (2015)
20. Tong, Y., She, J., Ding, B., Chen, L., Wo, T., Xu, K.: Online minimum matching in real-time spatial data: experiments and analysis. Proceedings of the VLDB Endowment 9(12), 1053–1064 (2016)
21. Tong, Y., She, J., Ding, B., Wang, L., Chen, L.: Online mobile micro-task allocation in spatial crowdsourcing. In: ICDE. pp. 49–60 (2016)
22. Tong, Y., She, J., Meng, R.: Bottleneck-aware arrangement over event-based social networks: the max-min approach. World Wide Web: Internet and Web Information Systems 19(6), 1151–1177 (2016)