Top-k Team Recommendation in Spatial Crowdsourcing

Dawei Gao[1], Yongxin Tong[1](✉), Jieying She[2], Tianshu Song[1], Lei Chen[2], and Ke Xu[1]

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

Abstract. With the rapid development of Mobile Internet and Online To Offline (O2O) marketing model, various spatial crowdsourcing platforms, such as Gigwalk and Gmission, are getting popular. Most existing studies assume that spatial crowdsourced tasks are simple and trivial. However, many real crowdsourced tasks are complex and need to be collaboratively finished by a team of crowd workers with different skills. Therefore, an important issue of spatial crowdsourcing platforms is to recommend some suitable teams of crowd workers to satisfy the requirements of skills in a task. In this paper, to address the issue, we first propose a more practical problem, called *Top-k Team Recommendation in spatial crowdsourcing* (Top$k$TR) problem. We prove that the Top$k$TR problem is NP-hard and design a two-level-based framework, which includes an approximation algorithm with provable approximation ratio and an exact algorithm with pruning techniques to address it. Finally, we verify the effectiveness and efficiency of the proposed methods through extensive experiments on real and synthetic datasets.

1 Introduction

Recently, thanks to the development and wide use of smartphones and mobile Internet, the studies of crowdsourcing are switching from traditional crowdsourcing problems [15,16] to the issues in spatial crowdsourcing markets, such as Gigwalk, Waze, Gmission, etc., where crowd workers (workers for short in this paper) are paid to perform spatial crowsourced tasks (tasks for short in this paper) that are requested on a mobile crowdsourcing platform [17].

Most existing studies on spatial crowdsourcing mainly focus on the problems of task assignment [6,7,13,14,17], which are to assign tasks to suitable workers, and assume that tasks are all simple and trivial. However, in real applications, there are many complex spatial crowdsourced tasks, which often need to be collaboratively completed by a team of crowd workers with different skills. Imagine the following scenario. David is a social enthusiast and usually organizes different types of parties on weekends. On the coming Saturday, he intends to hold

© Springer International Publishing Switzerland 2016
B. Cui et al. (Eds.): WAIM 2016, Part I, LNCS 9658, pp. 191–204, 2016.
DOI: 10.1007/978-3-319-39937-9_15

Table 1. The skill, payoff and capacity information of crowd workers

	w_1	w_2	w_3	w_4	w_5
Skills	$\{e_1, e_2\}$	$\{e_1\}$	$\{e_2, e_3\}$	$\{e_2\}$	$\{e_1, e_2, e_3\}$
Price	2	1	3	1	2
Capacity	1	1	2	1	1

a dance party and needs to recruit some sound engineers, guitarists, cooks and dancers. However, David faces a dilemma that his limited budget cannot afford to recruit all the aforementioned workers. He has to recruit fewer cheap crowd workers who have multiple skills and can take up several responsibilities, e.g. a worker can play the guitar and also manage the sound systems. Therefore, David posts his tasks on a spatial crowdsourcing platform, Gigwalk, and wants to find cheap crowd workers to satisfy his requirements. In fact, many task requestors have the same appeal: *can spatial crowdsourcing platforms recommend several cheaper candidate teams of crowd workers who can satisfy the multiple skills requirement of the tasks?* To further illustrate this motivation, we go through a toy example as follows.

Example 1. Suppose we have five crowd workers $w_1 - w_5$ on a spatial crowdsourcing platform, whose locations are shown in a 2D space (X, Y) in Fig. 1. Each worker owns different skills, which are shown in the second row in Table 1. Furthermore, each worker has a price for each task and a capacity, which is the maximum number of skills that can be used in a task that he/she performs, which are presented in the third and forth rows in Table 1. Moreover, a team-oriented spatial crowdsourced task and its locality range (the dotted circle) are shown in Fig. 1. Particularly, the task requires that the recruited crowd workers must cover three skills, $\{e_1, e_2, e_3\}$. To help the task requestor save cost, the spatial crowdsourcing platform usually recommends top-k cheapest teams of

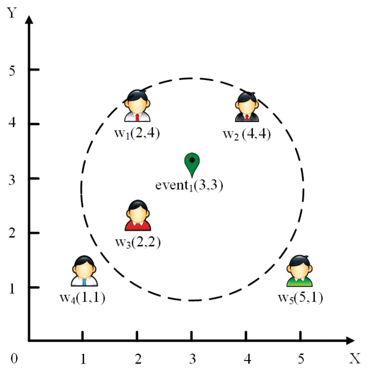


Fig. 1. Locations of the task and the five crowd workers

crowd workers, who can satisfy the requirement of skills. Furthermore, the recommended teams should not have free riders. In other words, each recommended team cannot satisfy the required skills if any worker in the team leaves. Therefore, in this example, the top-2 cheapest teams without free riders are $\{w_2, w_3\}$ and $\{w_1, w_3\}$, respectively, if the parameter $k = 2$.

As discussed above, we propose a novel team recommendation problem in spatial crowdsourcing, called the *top-k team recommendation in spatial crowdsourcing* (Top$k$TR) problem. As the example above indicates, the Top$k$TR problem not only recommends k cheapest teams but also satisfies the constraints of spatial range and skill requirement of tasks, capacity of workers, and no free rider in teams. Notice that the Top-1TR problem can be reduced to the classical team formation problem if the constraints on the capacity of workers and free riders are removed. More importantly, the Top$k$TR problem needs to return k teams instead of the cheapest team, which is its main challenge. We make the following contributions.

- We identify a new type of team-oriented spatial crowdsourcing applications and formally define it as the top-k team recommendation in spatial crowdsourcing (Top$k$TR) problem.
- We prove that the Top$k$TR problem is NP-hard and design a two-level-based framework, which not only includes an exact algorithm to provide the exact solution but also can seamlessly integrate an approximation algorithm to guarantee $\ln |E_t|$ theoretical approximation ratio, where $|E_t|$ is the number of required skills of the task.
- We verify the effectiveness and efficiency of the proposed methods through extensive experiments on real and synthetic datasets.

The rest of the paper is organized as follows. In Sect. 2, we formally define our problem and prove its NP-hardness. In Sect. 3, we present an two-level-based framework and its exact and approximation solutions. Extensive experiments on both synthetic and real datasets are presented in Sect. 4. We review related works and conclude this paper in Sects. 5 and 6, respectively.

2 Problem Statement

We formally define the *<u>Top-k</u> <u>T</u>eam <u>R</u>ecommendation in spatial crowdsourcing* (Top$k$TR) problem and prove that this problem is NP-hard. For convenience of discussion, we assume $E = <e_1, \cdots, e_m>$ to be a universe of m skills.

Definition 1 (Team-oriented Spatial Crowdsourced Task). *A team-oriented spatial crowdsourced task ("task" for short), denoted by* $t = <\boldsymbol{l}_t, E_t, r_t>$, *at location* $\boldsymbol{l}_t$ *in a 2D space is posted to the crowd workers, who are located in the circular range with the radius* r_t *around* $\boldsymbol{l}_t$, *on the platform. Furthermore,* $E_t \subseteq E$ *is the set of the required skills of the task* t *for the recruited team of crowd workers.*

Definition 2 (Crowd Worker). *A crowd worker ("worker" for short) is denoted by* $w = <\boldsymbol{l}_w, E_w, p_w, c_w>$*, where* $\boldsymbol{l}_w$ *is the location of the worker in a 2D space,* $E_w \subseteq E$ *is the set of skills that the worker is good at,* p_w *is the payoff for the worker to complete a crowdsourced task, and* c_w *is the capacity of the worker, namely the maximum number of skills used by the worker to complete a crowdsourced task.*

Note that the team-oriented spatial crowdsourced tasks studied in this paper, e.g. organizing a party, renovating a room, etc., usually need to be completed in teams. Though a worker may be good at multiple required skills, he/she cannot finish all the works by himself/herself. Therefore, we limit the capacity of each worker to balance the workload of the whole team. To simplify the problem, we assume that each worker receives the same payoff for different tasks since the capacity of the used skills of each user can be restricted. On one hand, these workers often have similar workloads and do not need a team leader to do a task. On the other hand, our model can be also easily extended to address the scenario where workers ask for different rewards for his/her different skills. Finally, we define our problem as follows.

Definition 3 (Top$k$TR Problem). *Given a team-oriented spatial crowdsourced task* t*, a set of crowd workers* W*, and the number of recommended crowdsourced teams* k*, the Top*k*TR problem is to find* k *crowdsourced teams,* $\{g_1, \cdots, g_k\}$ $(\forall g_i \subseteq W, 1 \leq i \leq k)$ *with* k *minimum* $Cost(g_i) = \sum_{w \in g_i} p_w$ *such that the following constraints are satisfied:*

- *Skill constraint: each required skill is covered by the skills of at least one worker.*
- *Range constraint: each worker* $w \in g_i$ *must locate in the restricted range of the task* t*.*
- *Capacity constraint: the number of skills used by each worker* $w \in g_i$ *cannot exceed* w*'s capacity* c_w*.*
- *Free-rider constraint: no team still satisfies the skill constraint if any worker in the team leaves.*

Theorem 1. *The Top*k*TR Problem is NP-hard.*

Proof. When $k = 1$ and the capacity constraint is ignored, such special case of the Top$k$TR problem is equivalent to the team formation problem [8], which has been proven to be NP-hard. Therefore, the Top$k$TR problem is also an NP-hard problem.

3 A Two-Level-Based Framework

To solve the problem effectively, we present a two-level-based algorithm framework. The first level aims to find the current top-1 feasible team with the minimum price, and the second level utilizes the function in the first level to iteratively maintain the top-k best teams. Particularly, the two-level-based framework has a nice property that the whole algorithm can keep the same approximation guarantee of the algorithm as in the first level.

Algorithm 1. Two-Level-based Framework

```
input : $W = \{w_1, \cdots, w_{|W|}\}, t, k$, and top-1 function top-1(.,.)
output: Top-$k$ teams $G = \{g_1, \cdots, g_k\}$.
1  $Queue \leftarrow \varnothing$; $G \leftarrow \varnothing$;
2  Insert the team generated by the function top-1(W,t) into $Queue$;
3  while $Queue \neq \varnothing$ do
4      $res \leftarrow$ top of $Queue$;
5      $G \leftarrow G \bigcup \{res\}$;
6      if $|G| = k$ then
7          return $G$;
8      Remove top of $Queue$;
9      foreach $w \in res$ do
10         Insert the team generated by the function top-1($W_{res} - \{w\}$,t) into $Queue$;
```

3.1 Overview of the Framework

The main idea of the two-level framework is that the top-2 best team can be discovered if and only if the top-1 best team is found first. In other words, after excluding the top-1 best team from the solution space, not only the size of the solution space is shrunken, but also the global top-2 best team must be the local top-1 best team in the shrunken solution space. The function of finding the local top-1 best team is denoted as the top-1 function in the first level, which will be described in details as the approximation algorithm and the exact algorithm in Sects. 3.1 and 3.2, respectively.

The framework is shown in Algorithm 1. We first initialize an empty priority queue of teams $Queue$, which sorts the elements in non-increasing prices of the teams, and the top-k teams G in lines 1–2. In line 3, we use a given algorithm, which can be exact or approximate, to get the exact or approximate top-1 team and insert it into $Queue$. In lines 4–11, if $Queue$ is not empty, we get the top element res of $Queue$ and insert res into G. For each w in res, we reduce the solution space of res to $W_{res} - \{w\}$, find a solution in it, and insert the solution into $Queue$. We repeat this procedure until we get k teams.

As introduced above, the framework has a nice property that the whole algorithm can keep the same approximation guarantee of the algorithm (top-1 function) in the first level.

Theorem 2. *If the top-1 function top-1(.,.) in the framework is an approximation algorithm with approximate ratio of r, the approximate cost of the i-th team in the approximation top-k teams by the framework keeps the same approximate ratio compared to the cost of the corresponding i-th exact team.*

Proof. We represent the approximation top-k teams generated by the framework as $\{g_1^a, \cdots, g_k^a\}$, and the exact top-k teams is denoted as $\{g_1^{ex}, \cdots, g_k^{ex}\}$. Because the top-1 function top-1(.,.) has approximate ratio of r, Cost$(g_1^a) \leq r\times$ Cost(g_1^{ex}). When the framework excludes g_1^a from the solution space and utilizes the top-1 function to obtain the other local top-1 team, it has the following two cases: (1) if $g_1^a = g_1^{ex}$, we have $g_2^a \leq r \times g_2^{ex}$; (2) $g_1^a \neq g_1^{ex}$, $g_2^a \leq r \times g_1^{ex}$.

Algorithm 2. Top-1 Greedy Approximation Algorithm

```
input  : $W = \{w_1, \cdots, w_{|W|}\}, t$
output: Team $g$.
1 $g \leftarrow \varnothing$;
2 while the team $g$ cannot satisfy the requirement of $E_t$ do
3     $w \leftarrow argmax_{w \in W}(\frac{MAXITEM(g \bigcup \{w\}) - MAXITEM(g)}{p_w})$;
4     $g \leftarrow g \bigcup \{w\}$;
5 return $Refine(g)$
```

3.2 Top-1 Approximation Algorithm

The main idea of the top-1 approximation algorithm utilizes the greedy strategy to choose the best worker w, who can bring the maximum gain to the current partial team g. Algorithm 2 illustrates the top-1 approximation algorithm. We first initialize a empty team g in line 1. In lines 2–4, when g cannot satisfy the requirement of skills of the task t, denoted by E_t, the algorithm selects a worker w with the maximum gain and the least price for the current team. The function $MAXITEM(.)$ is used to calculate the number of skills in E_t that can be covered by a specific team. In line 5, since g may contain free-rider workers, we have to refine the team.

Example 2. Back to our running example in Example 1. The running process of the top-1 approximation algorithm is shown in Table 2. In the first round, we choose w_2 with the biggest benefit 1. Since $\{w_2\}$ can not handle the task, we proceed to choose w_3 with the biggest benefit of $\frac{2}{3}$. Now, we can handle the task with $\{w_2, w_3\}$ and the price is 4.

Approximation Ratio. The approximation ratio of the algorithm is $O(\ln|E_t|)$. Inspired by [10], it is easy to get the approximation ratio of Algorithm 2. Due to the limited space, the details of the approximation ratio proof are omitted in this paper.

Table 2. The running process of Top-1 approximation algorithm

Round	w_1	w_2	w_3
1	1/2	**1**	2/3
2	1/2		**2/3**

Complexity Analysis. The time consumed by $MAXITEM$ is $O(|E_t|^2 \log(|E_t|))$. Line 3 will be executed at most $|E_t|$ times. The Refine step takes $O(2^{|E_t|})$ time. Thus, the total time complexity is $O(|W||E_t|^3 \log(|E_t|) + 2^{|E_t|})$. Since $|E_t|$ is usually very small in real applications, the algorithm is still efficient.

Finally, the following example illustrates the whole process of the complete approximation algorithm based on the two-level-based framework.

Algorithm 3. Top-1 Exact Algorithm

```
   input  : $W = \{w_1, \cdots, w_{|W|}\}, t$
   output: Team $g$.
1  $C_g \leftarrow$ Price of Top-1 Greedy Approximation Algorithm($W, t$);
2  $state \leftarrow \varnothing$;
3  foreach $w \in W$ do
4      if $p_w \leqslant C_g$ then
5          foreach cover condition of $w$ as $c$ do
6              foreach $s \in state$ do
7                  if $s.P + c.P \leqslant C_g$ then
8                      Insert new cover condition of $s + c$ into $temp\_state$;
9                  Insert c into $temp\_state$;
10             update $state$ using $temp\_state$ and clear $temp\_state$;
11 $T \leftarrow$ cover condition of skills $E_t$;
12 return $T$;
```

Example 3. Back to our running example in Example 1. Suppose $k = 2$ and the required skills of the task $t = \{1, 2, 3\}$. We first use the Top-1 greedy approximation algorithm to get a team of $\{w_1, w_3\}$ in the first level of the framework. Then we continue to adopt the Top-1 greedy approximation algorithm to find the local top-1 teams from $W - \{w_1\}$ and $W - \{w_3\}$. The returned teams are $\{w_1, w_3\}$ and $\varnothing$ respectively. Thus, the final top-2 teams generated by the whole framework are $\{w_2, w_3\}$ and $\{w_1, w_3\}$.

3.3 Top-1 Exact Algorithm

Since the number of skills required by a task is often not large, the main idea of the Top-1 exact algorithm is to enumerate the cover state of every proper subset of the intersection of the skills between a worker and a task. For each proper subset, we maintain a cover state of the covered skills and the total price of workers. We update the global cover state when processing each worker. When we have processed all the workers, the cover states of all the required skills of the task are the exact solution.

The exact algorithm is shown in Algorithm 3. We first get a approximate solution using a greedy algorithm and store the price of the solution in C_g in line 1. We then initialize *state* to store the currently best cover state. In lines 4–10, we successively process each worker in W. For worker w, if w_p is not larger than C_g, we enumerate all the cover states of w_p. For each cover state c, we combine it with cover state *state*. If the combined price is not larger than C_g, we store the current cover state in *temp_state*. We finally store c in *temp_state* and use it to update *state*. After we have processed all the workers in W, we check the cover state of the required skills of task t and its associated team is the best team. In line 4 and line 7, we adopt two pruning strategies. In line 4, we use C_g to prune a single worker whose price is too high. In line 7, we use C_g to prune a new cover state whose price is too high.

Example 4. Back to our running example in Example 1. We first use the top-1 approximation algorithm shown in Algorithm 2 to get an approximate solution $T = \{w_2, w_3\}$ with total price of 4, which is used as the current lower bound. Then we maintain the cover state using a triple structure, which contains the covered skills, the workers and the total price of the current optimal team for each possible combination of the required skills. w_1 can cover skill 1 or 2 with price 2, which is less than the lower bound of 4, so the cover state of w_1 can be $\{<\{1\}, \{w_1\}, 2>, <\{2\}, \{w_1\}, 2>\}$. As w_1 is the first worker we process, we just update the current best cover state as $\{<\{1\}, \{w_1\}, 2>, <2, \{w_1\}, 2>\}$. We then proceed to process w_2. We combine the only cover state, $<\{1\}, \{w_2\}, 1>$ with the cover states in *state*, and then we get a new cover state of $<\{1, 2\}, \{w_1, w_2\}, 2>$. After processing w_2, the current best cover state is $\{<\{1\}, \{w_2\}, 1>, <\{2\}, \{w1\}, 2>, <\{1, 2\}, \{w_1, w_2\}, 2>\}$. We can process w_3 similarly and the final cover state is $\{<\{1\}, \{w_2\}, 1>, <\{2\}, \{w_1\}, 2>, <\{1, 2\}, \{w_1, w_2\}, 2>, <\{1, 2, 3\}, \{w_2, w_3\}, 4>\}$ and the best team is $\{w_2, w_3\}$.

Complexity Analysis. Line 3 runs $|W|$ times, line 5 runs $C(|E_t|, |E_t|/2)$ times, and line 8 runs $2^{|E_t|}$ times. Therefore, the total time complexity is $O(|W|(2^{|E_t|}))$. When $|E_t|$ is not too large, the exact algorithm can be used.

4 Experimental Study

4.1 Experimental Setup

We use a real dataset collected from gMission [5], which is a research-based general spatial crowdsourcing platform. In the gMission dataset, every task has a task description, a location, a radius of the restricted range, and the required skills. Each worker is also associated with a location, a set of his/her owning skills, a price, and a capacity of skills that s/he completes a task. Currently, users often recruit crowd workers to organize all kinds of activities on the gMission platform. In this paper, our real dataset includes the information of 11205 crowd workers, where the average number of skills and the average capacity owned by the workers are 5.46 and 4.18, respectively. We also use synthetic dataset for evaluation. In the synthetic dataset, the capacity and the number of skills owned by a worker follow uniform distribution in the range of 1 to 20, respectively. Statistics of the synthetic dataset are shown in Table 3, where we mark our default settings in bold font.

Based on the two-level-based framework, we evaluate an approximation algorithm (Algorithms 1 and 2), called TTR-Greedy, and two exact algorithms (Algorithms 1 and 3), called TTR-Exact (which does not use the proposed pruning rules) and TTR-ExactPrune, and a baseline algorithm in terms of total utility score, running time and memory cost, and study the effect of varying parameters on the performance of the algorithms. The baseline algorithm uses a simple random greedy strategy, which first finds the best team, then randomly removes

Table 3. Synthetic Dataset

Factor	Setting
$\|W\|$	1000, 3000, **5000**, 7000, 9000
k	4, **8**, 12, 16, 20
$\|E_t\|$	4, **8**, 12, 16, 20
$\mu_{\|E_w\|}$	2, 4 **6**, 8, 10
$\sigma_{\|}E_w\|$	8, 10, **12**, 14, 16
Scalability ($\|W\|$)	10K, 30K, 50K, 70K, 90K

a worker from the best team from the set of workers, and iteratively finds the other $k-1$ best teams following the two steps above. The algorithms are implemented in Visual C++ 2010, and the experiments were performed on a machine with Intel(R) Core(TM) i5 2.40 GHz CPU and 4GB main memory.

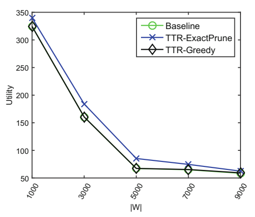
4.2 Experiment Results

In this subsection, we test the performance of our proposed algorithms through varying different parameters.

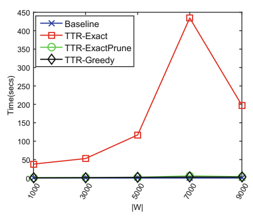
Effect of Cardinality of *W*. The results of varying $|W|$ are presented in Fig. 2a to c. Since TTR-Exact and TTR-ExactPrune return the same utility results, only utility results of TTR-ExactPrune are plotted. We can first observe that the utility decreases as $|W|$ increases, which is reasonable as more high-quality workers can are available. Also, we can see that TTR-Greedy is nearly as good as the exact algorithms. As for running time, TTR-Exact consumes more time with more workers due to larger search space while the TTR-ExactPrune is quite efficient due to its pruning techniques. The other algorithms do not vary much in running time. For memory, TTR-ExactPrune is the most efficient while TTR-Exact and TTR-Greedy are less efficient.

Effect of Parameter *k*. The results of varying k are presented in Fig. 2d to f. We can observe that the utility, running time and memory generally increase as k increases, which is reasonable as more teams need to be recommended. Again, we can see that TTR-Greedy is nearly as good as the exact algorithms but runs much faster. Also, we can see that the pruning techniques are quite effective as TTR-ExactPrune is much faster than TTR-Exact. Finally, TTR-Greedy is the most inefficient in terms of memory consumption.

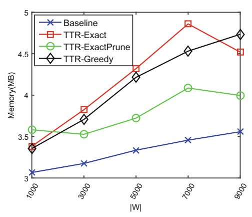
Effect of the Number of Required Skills in Tasks. The results are presented in Fig. 2g to i. We can see that the utility values increase first with increasing number of required skills $|E_t|$ but decrease later when $|E_t|$ further increases. The possible reason is that when $|E_t|$ is not large, the required skills are still quite diverse and thus more workers need to be hired to complete the task as $|E_t|$ increases. However, as $|E_t|$ becomes too large, many workers may



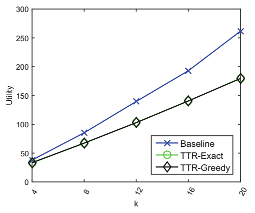
(a) Utility of varying $|W|$



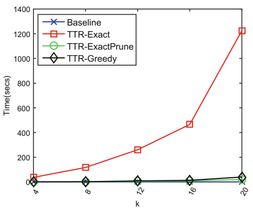
(b) Time of varying $|W|$



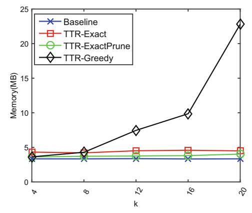
(c) Memory of varying $|W|$



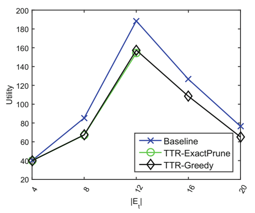
(d) Utility of varying k



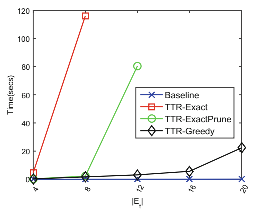
(e) Time of varying k



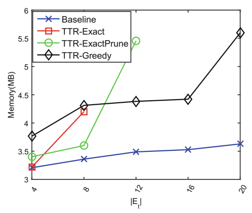
(f) Memory of varying k



(g) Utility of varying $|E_t|$



(h) Time of varying $|E_t|$



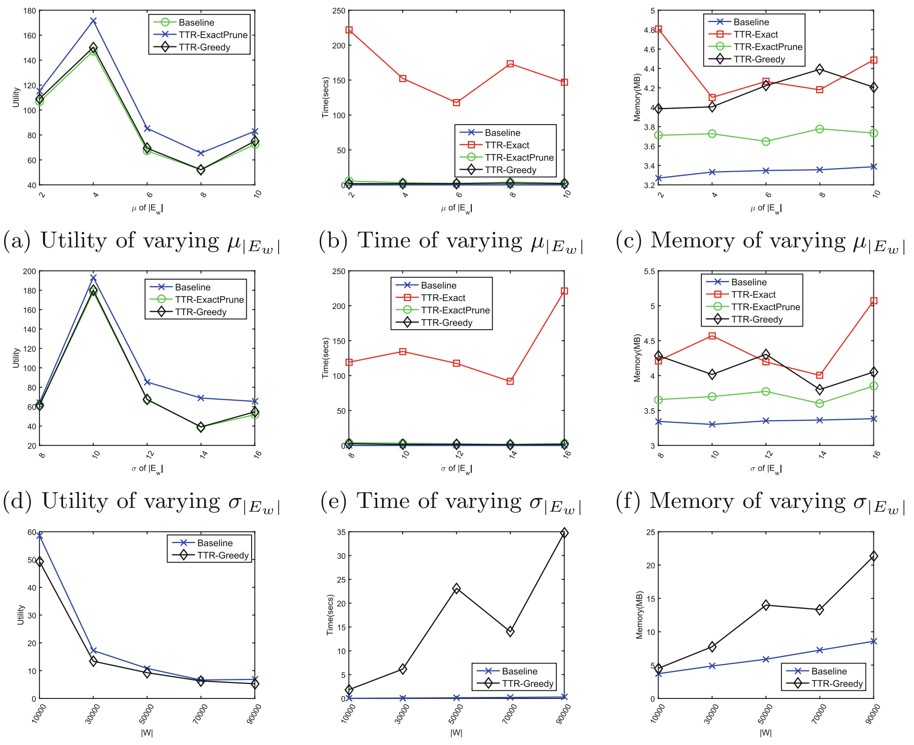
(i) Memory of varying $|E_t|$

Fig. 2. Results on varying $|W|$, k, and $|E_t|$.

use their own multiple skills to complete the task and thus less workers may be needed. As for running time and memory, we can observe that the values generally increase. Again, TTR-ExactPrune is highly inefficient compared with the other algorithms. Notice that the exact algorithms run very long time when $|E_t|$ is large, so we do not plot their results when $|E_t|$ is larger than 12.

Effect of the Distribution of the Number of Skills Per Each Worker (μ and σ). The results are presented in Fig. 3a to f. We can first observe that the utility value first increases as μ and σ increase and then drops when μ and σ further increase. The possible reason is that when μ and σ first increase, the skills of workers are more diverse and may not cover the requirements of the tasks and thus more workers are still needed. However, as μ and σ further increase, many workers can utilize their multiple skills and thus less workers are needed. As for running time, TTR-Exact is again very inefficient. Finally, for memory, TTR-ExactPrune is more efficient than TTR-Exact and TTR-Greedy.

Scalability. The results are presented in Fig. 3g to i. Since the exact algorithms are not efficient enough, we only study the scalability of TTR-Greedy. We can



(a) Utility of varying $\mu_{|E_w|}$ (b) Time of varying $\mu_{|E_w|}$ (c) Memory of varying $\mu_{|E_w|}$

(d) Utility of varying $\sigma_{|E_w|}$ (e) Time of varying $\sigma_{|E_w|}$ (f) Memory of varying $\sigma_{|E_w|}$

(g) Utility of scalability test (h) Time of scalability test (i) Memory of scalability test

Fig. 3. Results on varying $\mu_{|E_w|}$, $\sigma_{|E_w|}$, and scalability test.

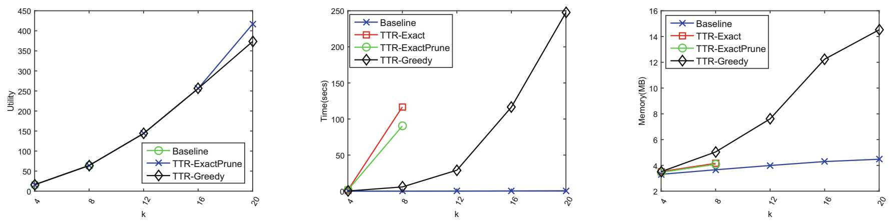
see that the running time and memory consumption TTR-Greedy is still quite small when the scale of data is large.

Real Dataset. The results on real dataset are shown in Fig. 4a to c, where we vary k. We can observe similar patterns as those in Fig. 2d to f. Notice that the exact algorithms are not efficient enough on the dataset, so no result of them when k is larger than 8 is presented.

Conclusion. For utility, TTR-Greedy is nearly as good as the exact algorithms, and TTR-Greedy and the exact algorithms all perform better than the baseline algorithm do. As for running time, TTR-Exact is the most inefficient, while TTR-ExactPrune is much more efficient than TTR-Exact due to its pruning techniques but is still slower than TTR-Greedy.

5 Related Work

In this section, we review related works from two categories, spatial crowdsourcing and team formation.



(a) Utility in real data (b) Time in real data (c) Memory in real data

Fig. 4. Performance on the real dataset.

5.1 Spatial Crowdsourcing

Most works on spatial crowdsourcing study the task assignment problem. [6,14] aim to maximize the number of tasks that are assigned to workers. Furthermore, the conflict-aware spatial task assignment problems are studied [11,12,18]. Recently, the issue of online task assignment in dynamic spatial crowdsourcing scenarios is proposed [17]. [7] further studies the reliability of crowd workers based on [6]. [13] studies the location privacy protection problem for the workers. [7] studies the route planning problem for a crowd worker and tries to maximize the number of completed tasks. The corresponding online version of [7] is studied in [9]. Although the aforementioned works study the task allocation problem on spatial crowdsourcing, they always assume that spatial crowdsourcing tasks are simple micro-tasks and ignore that some real spatial crowdsourced tasks often need to be collaboratively completed by a team of crowd workers.

5.2 Team Formation Problem

Another closely related topic is the team formation problem [8], which aims to find the minimum cost team of experts according to skills and relationships of users in social networks. [1,2] further studies the workload balance issue in the static and dynamic team formation problem. The capacity constraint of experts is also considered as an variant of the team formation problem in [10]. Moreover, the problems of discovering crowed experts in social media market are also studied [3,4]. The above works only consider to find the minimum cost team, namely top-1 team, instead of top-k teams without free riders. In addition, we address the spatial scenarios rather than the social networks scenarios.

6 Conclusion

In this paper, we study a novel spatial crowdsourcing problem, called the *Top-k Team Recommendation in spatial crowdsourcing* (Top$k$TR), which is proven to be NP-hard. To address this problem, we design a two-level-based framework, which not only includes an exact algorithm with pruning techniques to get the exact solution but also seamlessly integrates an approximation algorithm to guarantee theoretical approximation ratio. Finally, we conduct extensive experiments which verify the efficiency and effectiveness of the proposed approaches.

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, the Hong Kong RGC Project N_HKUST637/13, NSFC Guang Dong Grant No. U1301253, Microsoft Research Asia Gift Grant, Google Faculty Award 2013.

References

1. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Power in unity: forming teams in large-scale community systems. In: CIKM 2010, pp. 599–608 (2010)
2. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Online team formation in social networks. In: WWW 2012, pp. 839–848 (2012)
3. Cao, C.C., She, J., Tong, Y., Chen, L.: Whom to ask?: jury selection for decision making tasks on micro-blog services. Proc. VLDB Endowment **5**(11), 1495–1506 (2012)
4. Cao, C.C., Tong, Y., Chen, L., Jagadish, H.V.: Wisemarket: a new paradigm for managing wisdom of online social users. In: SIGKDD 2013, pp. 455–463 (2013)
5. 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. Proc. VLDB Endowment **7**(14), 1629–1632 (2014)
6. Kazemi, L., Shahabi, C.: Geocrowd: enabling query answering with spatial crowdsourcing. In: GIS 2012, pp. 189–198 (2012)
7. Kazemi, L., Shahabi, C., Chen, L.: Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In: GIS 2013, pp. 304–313 (2013)
8. Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: SIGKDD 2009, pp. 467–476 (2009)
9. Li, Y., Yiu, M.L., Xu, W.: Oriented online route recommendation for spatial crowdsourcing task workers. In: Claramunt, C., Schneider, M., Wong, R.C.-W., Xiong, L., Loh, W.-K., Shahabi, C., Li, K.-J. (eds.) SSTD 2015. LNCS, vol. 9239, pp. 137–156. Springer, Heidelberg (2015)
10. Majumder, A., Datta, S., Naidu, K.: Capacitated team formation problem on social networks. In: SIGKDD 2012, pp. 1005–1013 (2012)
11. She, J., Tong, Y., Chen, L.: Utility-aware social event-participant planning. In: SIGMOD 2015, pp. 1629–1643 (2015)
12. She, J., Tong, Y., Chen, L., Cao, C.C.: Conflict-aware event-participant arrangement. In: ICDE 2015, pp. 735–746 (2015)
13. To, H., Ghinita, G., Shahabi, C.: A framework for protecting worker location privacy in spatial crowdsourcing. Proc. VLDB Endowment **7**(10), 919–930 (2014)
14. To, H., Shahabi, C., Kazemi, L.: A server-assigned spatial crowdsourcing framework. ACM Trans. Spat. Algorithms Syst. **1**(1), 2 (2015)
15. Tong, Y., Cao, C.C., Chen, L.: TCS: efficient topic discovery over crowd-oriented service data. In: SIGKDD 2014, pp. 861–870 (2014)

16. Tong, Y., Cao, C.C., Zhang, C.J., Li, Y., Chen, L.: Crowdcleaner: Data cleaning for multi-version data on the web via crowdsourcing. In: ICDE 2014, pp. 1182–1185 (2014)
17. Tong, Y., She, J., Ding, B., Wang, L., Chen, L.: Online mobile micro-task allocation in spatial crowdsourcing. In: ICDE 2016 (2016)
18. Tong, Y., She, J., Meng, R.: Bottleneck-aware arrangement over event-based social networks: the max-min approach. World Wide Web J. (to appear). doi:10.1007/s11280-015-0377-6