Task Assignment on Multi-Skill Oriented Spatial Crowdsourcing

Peng Cheng, Xiang Lian, Lei Chen, *Member, IEEE*, Jinsong Han, and Jizhong Zhao

Abstract—With the rapid development of mobile devices and crowdsourcing platforms, the spatial crowdsourcing has attracted much attention from the database community. Specifically, the spatial crowdsourcing refers to sending location-based requests to workers, based on their current positions. In this paper, we consider a spatial crowdsourcing scenario, in which each worker has a set of qualified skills, whereas each spatial task (e.g., repairing a house, decorating a room, and performing entertainment shows for a ceremony) is time-constrained, under the budget constraint, and required a set of skills. Under this scenario, we will study an important problem, namely *multi-skill spatial crowdsourcing* (MS-SC), which finds an optimal worker-and-task assignment strategy, such that skills between workers and tasks match with each other, and workers' benefits are maximized under the budget constraint. We prove that the MS-SC problem is NP-hard and intractable. Therefore, we propose three effective heuristic approaches, including greedy, g-divide-and-conquer and cost-model-based adaptive algorithms to get worker-and-task assignments. Through extensive experiments, we demonstrate the efficiency and effectiveness of our MS-SC processing approaches on both real and synthetic data sets.

Keywords—multi-skill spatial crowdsourcing, greedy algorithm, g-divide-and-conquer algorithm, cost-model-based adaptive algorithm

1 Introduction

With the popularity of GPS-equipped smart devices and wireless mobile networks [12], [17], nowadays people can easily identify and participate in some location-based tasks that are close to their current positions, such as taking photos/videos, repairing houses, and/or preparing for parties at some spatial locations. Recently, a new framework, namely *spatial crowdsourcing* [17], for employing workers to conduct spatial tasks, has emerged in both academia (e.g., the database community [9]) and industry (e.g., TaskRabbit [3]). A typical spatial crowdsourcing platform (e.g., gMission [9] and MediaQ [18]) assigns a number of moving *workers* to do *spatial tasks* nearby, which requires workers to physically move to some specified locations and accomplish these tasks.

Note that, not all spatial tasks are as simple as taking a photo or video clip (e.g., street view of Google Maps [2]), monitoring traffic conditions (e.g., Waze [4]), or reporting local hot spots (e.g., Foursquare [1]), which can be easily completed by providing answers via camera, sensing devices in smart phones, or naked eyes, respectively. In contrast, some spatial tasks can be rather complex, such as repairing a house, preparing for a party, and performing entertainment shows for a ceremony, which may consist of several steps/phases/aspects, and require demanding professional skills from workers. In other words, these complex tasks cannot be simply accomplished by normal workers, but require the skilled workers with specific expertise (e.g., fixing roofs or setting up the stage).

- *P. Cheng is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong, China, Email: pchengaa@cse.ust.hk.*
- *X. Lian is with the Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX 78539, USA, Email: xiang.lian@utrgv.edu.*
- *L. Chen is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong, China, Email: leichen@cse.ust.hk.*
- *J. Han and J. Zhao are with the Department of Computer Science, Xi'an Jiaotong University, Shaanxi, China. E-mails: {hanjinsong, zjz}@mail.xjtu.edu.cn.*

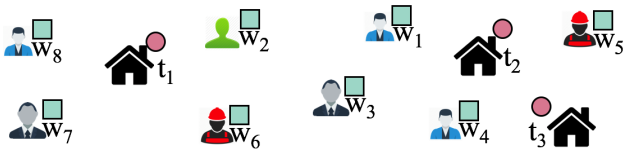


Fig. 1: An Example of Repairing a House in the Multi-Skill Spatial Crowdsourcing System.

TABLE 1: Worker/Task Skills

worker/task	skill key set
w_1, w_4, w_8	$\{a_1, a_4, a_6\}$
w_2	$\{a_5\}$
w_3, w_7	$\{a_2, a_3\}$
w_5, w_6	$\{a_1, a_5\}$
t_1, t_2, t_3	$\{a_1 \sim a_6\}$

TABLE 2: Descriptions of Skills

skill key	skill description
a_1	painting walls
a_2	repairing roofs
a_3	repairing floors
a_4	installing pipe systems
a_5	installing electronic components
a_6	cleaning

Inspired by the phenomenon of complex spatial tasks, in this paper, we will consider an important problem in the spatial crowdsourcing system, namely *multi-skill spatial crowdsourcing* (MS-SC), which assigns multi-skilled workers to those complex tasks, with the matching skill sets and high scores of the worker-and-task assignments.

In the sequel, we will illustrate the MS-SC problem by a motivation example of repairing a house.

Example (Repairing a House). *Consider a scenario of the spatial crowdsourcing in Figure 1, where a user wants to repair a house he/she just bought, in order to have a good living environment for his/her family. However, it is not an easy task to repair the house, which requires many challenging works (skills), such as repairing roofs/floors, replacing/installing pipe*

systems and electronic components, painting walls, and finally cleaning rooms. There are many skilled workers that can accomplish one or some of these skill types. In this case, the user can post a spatial task t_1, as shown in Figure 1, in the spatial crowdsourcing system, which specifies a set of required skills (given in Tables 1 and 2) for the house-repairing task, a deadline of the arrival time to repair, and the maximum budget that he/she would like to pay. In Figure 1, around the spatial location of task t_1, there are 8 workers, $w_1 \sim w_8$, each of whom has a different set of skills as given in Table 1. For example, worker w_1 has the skill set {painting walls, installing pipe systems, cleaning}. In addition, each worker has a maximum moving distance, as workers may not want to go to another city to conduct spatial tasks. Moreover, different workers also have different moving velocities.

To accomplish the spatial task t_1 (i.e., repair the house), the spatial crowdsourcing platform needs to select a best subset of workers w_i ($1 \leq i \leq 8$), such that the union of their skill sets can cover the required skill set of task t_1, and, moreover, workers can travel to the location of t_1 with the maximum net payment under the constraints of arrival times, workers' moving ranges, and budgets. For example, we can assign task t_1 with 3 workers w_2, w_7, and w_8, who are close to t_1, and whose skills can cover all the required skills of t_1.

Motivated by the example above, in this paper, we will formalize the MS-SC problem, which aims to efficiently assign workers to complex spatial tasks, under the task constraints of valid time periods and maximum budgets, such that the required skill sets of tasks are fully covered by those assigned workers, and the total score of the assignment (defined as the total profit of workers) is maximized.

Note that, existing works on spatial crowdsourcing focused on assigning workers to tasks to maximize the total number of completed tasks [17], the number of performed tasks for a worker with an optimal schedule [12], or the reliability-and-diversity score of assignments [10]. However, they did not take into account multi-skill covering of complex spatial tasks, time/distance constraints, and the assignment score with respect to task budgets and workers' salaries (excluding the traveling cost). Thus, we cannot directly apply prior solutions to solve our MS-SC problem.

In this paper, we first prove that our MS-SC problem in the spatial crowdsourcing system is NP-hard, by reducing it from the *Set Cover Problem* (SCP) [16]. As a result, the MS-SC problem is not tractable, and thus very challenging to achieve the optimal solution. Therefore, in this paper, we will tackle the MS-SC problem by proposing three effective heuristic approaches, greedy, *g-divide-and-conquer* (*g*-D&C), and cost-model-based adaptive algorithms, which can efficiently compute worker-and-task assignment pairs with the constraints/goals of skills, time, distance, and budgets.

Specifically, we make the following contributions.

- We formally define the *multi-skill spatial crowdsourcing* (MS-SC) problem in Section 2, under the constraints of multi-skill covering, time, distance, and budget for spatial workers/tasks in the spatial crowdsourcing system.
- We prove that the MS-SC problem is NP-hard, and thus intractable in Section 2.4.
- We propose efficient heuristic approaches, namely greedy, *g*-divide-and-conquer, and cost-model-based adaptive algorithms to tackle the MS-SC problem in Sections 4, 5, and 6, respectively.
- We conduct extensive experiments on real and synthetic data sets, and show the efficiency and effectiveness of our MS-SC approaches in Section 7.

Section 3 introduces a general framework for our MS-SC problem in spatial crowdsourcing systems. Section 8 reviews previous works on spatial crowdsourcing. Finally, Section 9 concludes this paper.

2 Problem Definition

In this section, we present the formal definition of the multi-skill spatial crowdsourcing, in which we assign multi-skilled workers with time-constrained complex spatial tasks.

2.1 Multi-Skilled Workers

We first define the multi-skilled workers in spatial crowdsourcing applications. Assume that $\Psi = \{a_1, a_2, ..., a_k\}$ is a universe of k abilities/skills. Each worker has one or multiple skills in Ψ, and can provide services for spatial tasks that require some skills in Ψ.

Definition 1: (Multi-Skilled Workers) Let $W_p = \{w_1, w_2, ..., w_n\}$ be a set of n multi-skilled workers at timestamp p. Each worker w_i ($1 \leq i \leq n$) has a set, X_i ($\subseteq \Psi$), of skills, is located at position $l_i(p)$ at timestamp p, can move with velocity v_i, and has a maximum moving distance d_i. ■

In Definition 1, the multi-skilled workers w_i can move dynamically with speed v_i in any direction, and at each timestamp p, they are located at spatial places $l_i(p)$, and prefer to move at most d_i distance from $l_i(p)$. They can freely join or leave the spatial crowdsourcing system. Moreover, each worker w_i is associated with a set, X_i, of skills, such as taking photos, cooking, and decorating rooms.

2.2 Time-Constrained Complex Spatial Tasks

Next, we define complex spatial tasks in the spatial crowdsourcing system, which are constrained by deadlines of arriving at task locations and budgets.

Definition 2: (Time-Constrained Complex Spatial Tasks) Let $T_p = \{t_1, t_2, ..., t_m\}$ be a set of time-constrained complex spatial tasks at timestamp p. Each task t_j ($1 \leq j \leq m$) is located at a specific location l_j, and workers are expected to reach the location of task t_j before the arrival deadline e_j. Moreover, to complete the task t_j, a set, Y_j ($\subseteq \Psi$), of skills is required for those assigned workers. Furthermore, each task t_j is associated with a budget, B_j, of salaries for workers. ■

As given in Definition 2, usually, a task requester creates a time-constrained spatial task t_j, which requires workers physically moving to a specific location l_j, and arriving at l_j before the arrival deadline e_j. Meanwhile, the task requester also specifies a budget, B_j, of salaries, that is, the maximum allowance that he/she is willing to pay for workers. This budget, B_j, can be either the reward cash or bonus points in the spatial crowdsourcing system.

Moreover, the spatial task t_j is often complex, in the sense that it might require several distinct skills (in Y_j) to be conducted. For example, a spatial task of repairing a house may require several skills, such as repairing floors, painting walls and cleaning.

2.3 The Multi-Skill Spatial Crowdsourcing Problem

In this subsection, we will formally define the multi-skill spatial crowdsourcing (MS-SC) problem, which assigns spatial tasks to workers such that workers can cover the skills required by tasks and the assignment strategy can achieve high scores.

Task Assignment Instance Set. Before we present the MS-SC problem, we first introduce the concept of task assignment instance set.

Definition 3: (Task Assignment Instance Set) At timestamp p, given a worker set W_p and a task set T_p, a *task assignment instance set*, denoted by I_p, is a set of worker-and-task assignment pairs in the form $\langle w_i, t_j \rangle$, where each worker $w_i \in W_p$ is assigned to at most one spatial task $t_j \in T_p$.

Moreover, we denote CT_p as the set of completed tasks t_j that can be reached before the arrival deadlines e_j, and accomplished by those assigned workers in I_p. ■

Intuitively, the task assignment instance set I_p is one valid worker-and-task assignment between worker set W_p and task set T_p. Each pair $\langle w_i, t_j \rangle$ is in I_p, if and only if this assignment satisfies the constraints of task t_j, with respect to distance (i.e., d_i), time (i.e., e_j), budget (i.e., B_j), and skills (i.e., Y_j).

In particular, for each pair $\langle w_i, t_j \rangle$, worker w_i must arrive at location l_j of the assigned task t_j before its arrival deadline e_j, and can support the skills required by task t_j, that is, $X_i \bigcap Y_j \neq \emptyset$. The distance between $l_i(p)$ and l_j should be less than d_i. Moreover, for all pairs in I_p that contain task t_j, the required skills of task t_j should be fully covered by skills of its assigned workers, that is, $Y_j \subseteq \cup_{\forall \langle w_i, t_j \rangle \in I_p} X_i$.

To assign a worker w_i to a task t_j, we need to pay him/her salary, c_{ij}, which is related to the travelling cost from the location, $l_i(p)$, of worker w_i to that, l_j, of task t_j. The travelling cost, c_{ij}, for vehicles can be calculated by the unit gas price per gallon times the number of gallons needed. For the public transportation, the cost c_{ij} can be computed by the fees per mile times the travelling distance. For walking, we can also provide the compensation fee for the worker with the cost c_{ij} proportional to his/her travelling distance.

Without loss of generality, we assume that the cost, c_{ij}, is proportional to the travelling distance, $dist(l_i(p), l_j)$, between $l_i(p)$ and l_j, where $dist(x, y)$ is a distance function between locations x and y. Formally, we have: $c_{ij} = C_i \cdot dist(l_i(p), l_j)$, where C_i is a constant (e.g., gas/transportation fee per mile).

Note that, for simplicity, in this paper, we use Euclidean distance as our distance function (i.e., $dist(x, y)$). We can easily extend our proposed approaches in this paper by considering other distance function (e.g., road-network distance), under the framework of the spatial crowdsourcing system.

The MS-SC Problem. In the sequel, we give the definition of our *multi-skill spatial crowdsourcing* (MS-SC) problem.

Definition 4: (Multi-Skill Spatial Crowdsourcing Problem) Given a time interval P, the problem of *multi-skill spatial crowdsourcing* (MS-SC) is to assign the available workers $w_i \in W_p$ to tasks $t_j \in T_p$, and to obtain a task assignment instance set, I_p, at each timestamp $p \in P$, such that:

1) any worker $w_i \in W_p$ is assigned to only one spatial task $t_j \in T_p$ such that his/her arrival time at location l_j before the arrival deadline e_j, the moving distance is less than the worker's maximum moving distance d_i, and all workers assigned to t_j have skill sets fully covering Y_j;
2) the total travelling cost of all the assigned workers to task t_j does not exceed the budget of the task, that is, $\sum_{\forall \langle w_i, t_j \rangle \in I_p} c_{ij} \leq B_j$; and
3) the total score, $\sum_{p \in P} S_p$, of the task assignment instance sets I_p within the time interval P is maximized,

where it holds that:

$$S_p = \sum_{t_j \in CT_p} B'_j, and \quad (1)$$

$$B'_j = B_j - \sum_{\langle w_i, t_j \rangle \in I_p} c_{ij}. \quad (2)$$

Definition 4 can be rewritten in the form of the linear programming problem below:

$$\begin{aligned}
\max \quad & \sum_{t_j \in CT_p} (B_j - \sum_{i=1}^{n} c_{ij} x_{ij}) \\
\text{s.t.} \quad & dist(l_j, l_i(p)) \leq (e_j - p) \cdot v_i \leq d_i && i = 1, \ldots, n; j = 1, \ldots, m, \\
& Y_j \subseteq \cup_{i=1}^{n} X_i \wedge x_{ij} && t_j \in CT_p, \\
& \sum_{i=1}^{n} c_{ij} x_{ij} \leq B_j && j = 1, \ldots, m, \\
& \sum_{j=1}^{m} x_{ij} \leq 1 && i = 1, ..., n,
\end{aligned}$$

where, x_{ij} is an indicator. If a worker w_i is assigned to a task t_j, $x_{ij} = 1$; otherwise, $x_{ij} = 0$.

In Definition 4, our MS-SC problem aims to assign workers w_i to tasks t_j such that: (1) workers w_i are able to reach locations, l_j, of tasks t_j on time and cover the required skill set Y_j, and the moving distance is less than d_i; (2) the total travelling cost of all the assigned workers should not exceed budget B_j; and (3) the total score, $\sum_{p \in P} S_p$, of the task-and-worker assignment within time interval P is maximized.

After the server-side assignment at a timestamp p, those assigned workers will become unavailable, and move to the locations of spatial tasks. Next, these workers will become available again, only if they finish/reject the assigned tasks.

Discussions on the Score S_p. Eq. (1) calculates the score, S_p, of a task-and-worker assignment by summing up *flexible budgets*, B'_j (given by Eq. (2)), of all the completed tasks $t_j \in CT_p$, where the *flexible budget* of task t_j is the remaining budget of task t_j after paying workers' travelling costs. Maximizing scores means maximizing the number of accomplished tasks while minimizing the travelling cost of workers.

Intuitively, each task t_j has a maximum budget B_j, which consists of two parts, the travelling cost of the assigned workers and the flexible budget. The former cost is related to the total travelling distance of workers, whereas the latter one can be freely and flexibly used for rewarding workers for their contributions to the task. Here, the distribution of the flexible budget among workers can follow existing incentive mechanisms in crowdsourcing [20], [23], which stimulate

workers who did the task better (i.e., with more rewards). We can reward the workers based on the requirement of the assigned tasks and the skills that they can provide, which is beyond the scope of this study. We would like to leave it as our future work.

Note that, in Eq. (1), the score, S_p, of the task assignment instance set I_p only takes into account those tasks that can be completed by the assigned workers (i.e., tasks in set CT_p). Here, a task can be completed, if the assigned workers can reach the task location before the deadline and finish the task with the required skills.

Since the spatial crowdsourcing system is quite dynamic, new tasks/workers may arrive at next timestamps. Thus, if we cannot find enough/proper workers to do the task at the current timestamp p, the task is still expected to be successfully assigned with workers and completed in future timestamps. Meanwhile, the task requester can be also informed by the spatial crowdsourcing system to increase the budget (i.e., with higher budget B_j, we can find more skilled candidate workers that satisfy the budget constraint). Therefore, in our definition of score S_p, we would only consider those tasks in CT_p that can be completed by the assigned workers at timestamp p, and maximize this score S_p.

2.4 The Hardness of the MS-SC Problem

With m time-constrained complex spatial tasks and n multi-skilled workers, in the worst case, there are an exponential number of possible task-and-worker assignment strategies, which leads to high time complexity, $O((m+1)^n)$. In this subsection, we prove that our MS-SC problem is NP-hard, by reducing a well-known NP-hard problem, *set cover problem* (SCP) [22], to the MS-SC problem.

Lemma 1: (Hardness of the MS-SC Problem) The problem of the Multi-Skill Spatial Crowdsourcing (MS-SC) is NP-hard.

Proof: Please refer to Appendix A in supplementary materials. □

Since the MS-SC problem involves multiple spatial tasks whose skill sets should be covered, we thus cannot directly use existing approximation algorithms for SCP (or its variants) to solve the MS-SC problem. What is more, we also need to find an assignment strategy such that workers and tasks match with each other (in terms of travelling time/cost, and budge constraints), which is more challenging.

Thus, due to the NP-hardness of our MS-SC problem, in subsequent sections, we will present a general framework for MS-SC processing and design 3 heuristic algorithms, namely greedy, g-divide-and-conquer, and cost-model-based adaptive approaches, to efficiently retrieve MS-SC answers.

Table 3 summarizes the commonly used symbols.

TABLE 3: Symbols and Descriptions.

Symbol	Description
T_p	a set of m time-constrained spatial tasks t_j at timestamp p
W_p	a set of n dynamically moving workers w_i at timestamp p
e_j	the deadline of arriving at the location of task t_j
$l_i(p)$	the position of worker w_i at timestamp p
l_j	the position of task t_j
X_i	a set of skills that worker w_i has
Y_j	a set of the required skills for task t_j
d_i	the maximum moving distance of worker w_i
B_j	the maximum budget of task t_j
I_p	the task assignment instance set at timestamp p
CT_p	a set of tasks that are assigned with workers at timestamp p and can be completed by these assigned workers
C_i	the unit price of the travelling cost of worker w_i
c_{ij}	the travelling cost from the location of worker w_i to that of task t_j
S_p	the score of the task assignment instance set I_p
ΔS_p	the score increase when changing the pair assignment

3 Framework for Solving the MS-SC Problem

In this section, we present a general framework, in Figure 2 for solving the MS-SC problem, which greedily assigns workers with spatial tasks for multiple rounds. For each round, at timestamp p, we first retrieve a set, T_p, of all the available spatial tasks, and a set, W_p, of available workers (lines 2-3). Here, the available task set T_p contains existing spatial tasks that have not been assigned with workers in the last round, and the ones that newly arrive at the system after the last round. Moreover, set W_p includes those workers who have accomplished (or rejected) the previously assigned tasks, and thus are available to receive new tasks in the current round.

In our spatial crowdsourcing system, we organize both sets T_p and W_p in a cost-model-based grid index. For the sake of space limitations, details about the index construction can be found in Appendix E of supplementary materials. Due to dynamic changes of sets T_p and W_p, we also update the grid index accordingly (line 4). Next, we utilize the grid index to efficiently retrieve a set, S, of valid worker-and-task candidate pairs (line 5). Note that, we only need to find the entire set of valid pairs at the beginning, and then update the set in subsequent timestamps, whose time cost is low with the help of our grid index. That is, we obtain those pairs of workers and tasks, $\langle w_i, t_j \rangle$, such that workers w_i can reach the locations of tasks t_j and satisfy the constraints of skill matching, time, and budgets for tasks t_j. With valid pairs in set S, we can apply our proposed algorithms, that is, *greedy* (GREEDY), *g-divide-and-conquer* (g-D&C), or *adaptive cost-model-based* (ADAPTIVE) approach, over set S, and obtain a good worker-and-task assignment strategy in an assignment instance set I_p, which is a subset of S (line 6). Finally, for each pair $\langle w_i, t_j \rangle$ in the selected worker-and-task assignment set I_p, we will notify worker w_i to do task t_j (lines 7-8).

Procedure MS-SC_Framework {
Input: a time interval P
Output: a worker-and-task assignment strategy within the time interval P
(1) for each timestamp p in P
(2) retrieve all the available spatial tasks to T_p
(3) retrieve all the available workers to W_p
(4) update the grid index for current T_p and W_p
(5) obtain a set, S, of valid worker-and-task pairs from the index
(6) use our *greedy*, *g-divide-and-conquer* or *adaptive cost-model-based* approach to obtain a good assignment instance set, $I_p \subseteq S$
(7) for each pair $\langle w_i, t_j \rangle$ in I_p
(8) inform worker w_i to conduct task t_j
}

Fig. 2: Framework for Solving the MS-SC Problem.

In particular, GREEDY selects a "best" worker-and-task pair that can achieve the maximum increase of the score ΔS_p (as given in Eq. (3)), which is a local optimal approach. g-D&C keeps dividing the problem into g subproblems on each level, until finally the number of tasks in each subproblem is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2550041, IEEE Transactions on Knowledge and Data Engineering

1 (which can be solved by the greedy algorithm). ADAPTIVE makes the trade-off between GREEDY and g-D&C, in terms of efficiency and accuracy, which adaptively decides the stopping level of the divide-and-conquer process.

4 The Greedy Approach

In this section, we will propose a greedy algorithm, which greedily selects one worker-and-task assignment, $\langle w_i, t_j \rangle$, at a time that can maximize the increase of the assignment score (i.e., $\sum_{\forall p \in P} S_p$ as given in Definition 4). This greedy algorithm can be applied in line 6 of the framework, MS-SC_Framework, in Fig. 2.

4.1 The Score Increase

Before we present the greedy algorithm, we first define the increase, ΔS_p, of score S_p (given in Eq. (1)), in the case where we assign a newly available worker w_i to task t_j. Specifically, from Eqs. (1) and (2), we define the score increase after assigning worker w_i to task t_j as follows:

$$\Delta S_p = S_p - S_{p-1} = \Delta B'_j = \frac{|X_i \cap (Y_j - \widetilde{Y_j})|}{|Y_j|} \cdot B_j - c_{ij}, \quad (3)$$

where $\widetilde{Y_j}$ is the set of skills that have been covered by those assigned workers (excluding the new worker w_i) for task t_j.

In Eq. (3), $\frac{|X_i \cap (Y_j - \widetilde{Y_j})|}{|Y_j|}$ is the ratio of skills for task t_j that have not been covered by (existing) assigned workers, but can be covered by the new worker w_i. Intuitively, the first term in Eq. (3) is the pre-allocated maximum budget based on the number of covered skills by the new worker w_i, whereas the second term, c_{ij}, is the travelling cost from location of w_i to that of t_j. Thus, the score increase, ΔS_p, in Eq. (3) is to measure the change of score (i.e., flexible budget) S_p, due to the assignment of worker w_i to task t_j.

4.2 Pruning Strategies

The score increase can be used as a measure to evaluate and decide which worker-and-task assignment pair should be added to the task assignment instance set I_p. That is, each time our greedy algorithm aims to choose one worker-and-task assignment pair in S with the highest score increase, which will be added to I_p (i.e., line 6 of MS-SC_Framework in Fig. 2). However, it is not efficient to enumerate all valid worker-and-task assignment pairs in S, and compute score increases. That is, in the worst case, the time complexity is as high as $O(m \cdot n)$, where m is the number of tasks and n is the number of workers. Therefore, in this subsection, we present three effective pruning methods (two for pruning workers and one for pruning tasks) to quickly filter out false alarms of worker-and-task pairs in set S.

The Worker-Pruning Strategy. When assigning available workers to tasks, we can rule out those valid worker-and-task pairs in S, which contain either *dominated* or *high-wage workers*, as given in Lemmas 2 and 3, respectively, below.

We say that a worker w_a is *dominated by* a worker w_b w.r.t. task t_j (denoted as $w_a \succ_{t_j} w_b$), if it holds that $X_a \subseteq X_b$ and $c_{aj} \geq c_{bj}$, where X_a and X_b are skill sets of workers w_a and w_b, and c_{aj} and c_{bj} are the travelling costs from locations of workers w_a and w_b to task t_j, respectively.

Lemma 2: (Pruning Dominated Workers) Given two worker-and-task pairs $\langle w_a, t_j \rangle$ and $\langle w_b, t_j \rangle$ in valid pair set S, if it holds that $w_a \succ_{t_j} w_b$, then we can safely prune the worker-and-task pair $\langle w_a, t_j \rangle$.

Proof: Please refer to Appendix B in supplementary materials. □

Lemma 2 indicates that if there exists a better worker w_b than worker w_a to do task t_j (in terms of both the skill set and the travelling cost), then we can safely filter out the assignment of worker w_a to task t_j.

Lemma 3: (Pruning High-Wage Workers) Let $\widetilde{c_{\cdot j}}$ be the total travelling cost for those workers that have already been assigned to task t_j. If the travelling cost c_{ij} of assigning a worker w_i to task t_j is greater than the remaining budget $(B_j - \widetilde{c_{\cdot j}})$ of task t_j, then we will not assign worker w_i to task t_j.

Proof: Please refer to Appendix C in supplementary materials. □

Intuitively, Lemma 3 shows that, if the wage of a worker w_i (including the travelling cost c_{ij}) exceeds the maximum budget B_j of task t_j (i.e., $c_{ij} > B_j - \widetilde{c_{\cdot j}}$), then we can safely prune the worker-and-task assignment pair $\langle w_i, t_j \rangle$.

The Task-Pruning Strategy. Let $W(t_j)$ be a set of valid workers that can be assigned to task t_j, and $\widetilde{W(t_j)}$ be a set of valid workers that have already been assigned to task t_j. We give the lemma of pruning those tasks with insufficient budgets below.

Lemma 4: (Pruning Tasks with Insufficient Budgets) If an unassigned worker $w_i \in (W(t_j) - \widetilde{W(t_j)})$ has the highest value of $\frac{\Delta S_p}{|X_i \cap (Y_j - \widetilde{Y_j})|}$, and the travelling cost, c_{ij}, of worker w_i exceeds the remaining budget $(B_j - \widetilde{c_{\cdot j}})$ of task t_j, then we can safely prune task t_j.

Proof: Please refer to Appendix D in supplementary materials. □

Intuitively, Lemma 4 provides the conditions of pruning tasks. That is, if any unassigned worker subset of $(W(t_j) - \widetilde{W(t_j)})$ either cannot fully cover the required skill set Y_j, or exceeds the remaining budget of task t_j, then we can directly prune all assignment pairs that contain task t_j.

To summarize, by utilizing Lemmas 2, 3 and 4, we do not have to check all worker-and-task assignments iteratively in our greedy algorithm. Instead, we can now apply our proposed three pruning methods, and effectively filter out those false alarms of assignment pairs, which can significantly reduce the number of times to compute the score increases.

4.3 The Greedy Algorithm

According to the definition of the score increase ΔS_p (as mentioned in Section 4.1), we propose a greedy algorithm, which iteratively assigns a worker to a spatial task that can always achieve the highest score increase.

Figure 3 shows the pseudo code of our MS-SC greedy algorithm, namely MS-SC_Greedy, which obtains one worker-and-task pair with the highest score increase each time, and

returns a task assignment instance set I_p with high score.

Initially, we set I_p to be empty, since no workers are assigned to any tasks (line 1). Next, we find out all valid worker-and-task pairs $\langle w_i, t_j \rangle$ in the crowdsourcing system at timestamp p (line 2). Here, the validity of pair $\langle w_i, t_j \rangle$ satisfies 4 conditions: (1) the distance between the current location, $l_i(p)$, of worker w_i and the location, l_j of task t_j is less than the maximum moving distance, d_i of worker w_i, that is, $dist(l_i(p), l_j) \leq d_i$; (2) worker w_i can arrive at the location, l_j, of task t_j before the arrival deadline e_j; (3) worker w_i have skills that task t_j requires; and (4) the travelling cost, c_{ij}, of worker w_i should not exceed the budget B_j of task t_j.

Then, for each round, we would select one valid worker-and-task assignment pair with the highest score increase, and add it to set I_p (lines 3-16). Specifically, in each round, we check every task t_j that is involved in valid pairs $\langle w_i, t_j \rangle$, and then prune those dominated and high-wage workers w_i, via Lemmas 2 and 3, respectively (lines 7-8). If worker w_i cannot be pruned by both pruning methods, then we add it to a candidate set S_{cand} for further checking (line 9). After obtaining all workers that match with task t_j, we apply Lemma 4 to filter out task t_j (if workers cannot be successfully assigned to t_j). If task t_j cannot be pruned, we will calculate the score increase, $\Delta S_p(w_i, t_j)$, for each pair $\langle w_i, t_j \rangle$ in S_{cand}; otherwise, we remove task t_j from task set T_p (lines 10-14).

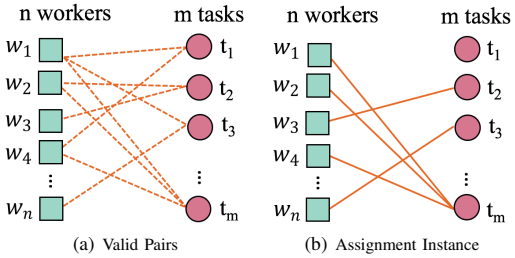
After we scan all tasks in T_p, we can retrieve one worker-and-task assignment pair, $\langle w_r, t_j \rangle$, from the candidate set S_{cand}, which has the highest score increase, and insert this pair to I_p (line 15). Since worker w_r has been assigned, we remove it from the worker set W_p (line 16). The process above repeats, until all workers have been assigned (i.e., $W_p = \emptyset$) or there are no tasks left (i.e., $T_p = \emptyset$) (line 3).

Procedure MS-SC_Greedy {
Input: n workers in W_p and m time-constrained spatial tasks in T_p
Output: a worker-and-task assignment instance set, I_p
(1) $I_p = \emptyset$;
(2) compute all valid worker-and-task pairs $\langle w_i, t_j \rangle$ from W_p and T_p
(3) while $W_p \neq \emptyset$ and $T_p \neq \emptyset$
(4) $S_{cand} = \emptyset$;
(5) for each task $t_j \in T_p$
(6) for each worker w_i in the valid pair $\langle w_i, t_j \rangle$
(7) if we cannot prune dominated worker w_i by Lemma 2
(8) if we cannot prune high-wage worker w_i by Lemma 3
(9) add $\langle w_i, t_j \rangle$ to S_{cand}
(10) if we cannot prune task t_j w.r.t. workers in S_{cand} by Lemma 4
(11) for each pair $\langle w_i, t_j \rangle$ w.r.t. task t_j in S_{cand}
(12) compute the score increase, $\Delta S_p(w_i, t_j)$
(13) else
(14) $T_p = T_p - \{t_j\}$
(15) obtain a pair, $\langle w_r, t_j \rangle \in S_{cand}$, with the highest score increase, $\Delta S_p(w_r, t_j)$, and add this pair to I_p
(16) $W_p = W_p - \{w_r\}$
(17) return I_p
}

Fig. 3: The MS-SC Greedy Algorithm.

Figure 4(a) illustrates an example of valid pairs, where n available workers and m spatial tasks are denoted by rectangular and circular nodes, respectively, and valid worker-and-task pairs are represented by dashed lines. Figure 4(b) depicts the result of one assignment with high score, where the bold lines indicate assignment pairs in I_p.

The Time Complexity. We next present the time complexity of the greedy algorithm, MS-SC_Greedy (in Figure 3). Specifically, the time cost of computing valid worker-and-task assignment pairs (line 2) is given by $O(m \cdot n)$ in the worst case, where any of n workers can be assigned to any of m tasks (i.e., $m \cdot n$ valid worker-and-task pairs). Then, for each round (lines 3-16), we apply pruning methods to $m \cdot n$ pairs, and select the pair with the highest score increase. In the worst case, pairs cannot be pruned, and thus the time complexity of computing score increases for these pairs is given by $O(m \cdot n)$. Moreover, since each of n workers can only be assigned to one spatial task, the number of iterations is at most n times. Therefore, the total time complexity of our greedy algorithm can be given by $O(m \cdot n^2)$.



(a) Valid Pairs (b) Assignment Instance

Fig. 4: Illustration of the Worker-and-Task Assignment.

5 The g-Divide-and-Conquer Approach

Although the greedy algorithm incrementally finds one worker-and-task assignment (with the highest score increase) at a time, it may incur the problem of only achieving local optimality. Therefore, in this section, we propose an efficient *g-divide-and-conquer algorithm* (g-D&C), which first divides the entire MS-SC problem into g subproblems, such that each subproblem involves a smaller subgroup of $\lceil m/g \rceil$ spatial tasks, and then conquers the subproblems recursively (until the final group size becomes 1). Since different numbers, g, of the divided subproblems may incur different time costs, in this paper, we will propose a novel cost-model-based method to estimate the best g value to divide the problem.

Specifically, for each subproblem/subgroup (containing $\lceil m/g \rceil$ tasks), we will tackle the worker-and-task assignment problem via recursion (note: the base case with the group size equal to 1 can be solved by the greedy algorithm [22], which has an approximation ratio of $\ln(N)$, where N is the total number of skills). During the recursive process, we combine/merge assignment results from subgroups, and obtain the assignment strategy for merged groups, by resolving the assignment conflicts among subgroups. Finally, we can return the task assignment instance set I_p, with respect to the entire worker and tasks sets.

In the sequel, we first discuss how to decompose the MS-SC problem into subproblems in Section 5.1. Then, we will illustrate our g-divide-and-conquer approach in Section 5.2, which utilizes the decomposition and merge (as will be discussed in Section 5.3) algorithms. Finally, we will provide a cost model in Section 5.4 to determine the best number g of subproblems during the g-D&C process.

5.1 MS-SC Problem Decompositions

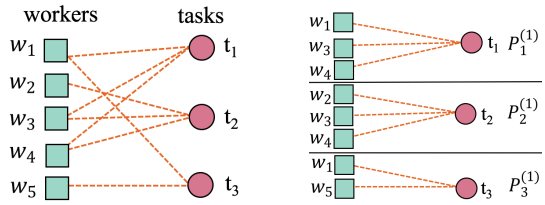
In this subsection, we discuss how to decompose a MS-SC problem into subproblems. In order to illustrate the decom-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2550041, IEEE Transactions on Knowledge and Data Engineering

position, we first convert our original MS-SC problem into a representation of a bipartite graph.

Bipartite Graph Representation of the MS-SC Problem. Specifically, given a worker set W_p and a spatial task set T_p, we denote each worker/task (i.e., w_i or t_j) as a vertex in the bipartite graph, where worker and task vertices have distinct vertex types. There exists an edge between a worker vertex w_i and a task vertex t_j, if and only if worker w_i can reach spatial task t_j under the constraints of skills (i.e., $X_i \cap Y_j \neq \emptyset$), time (i.e., arrival time is before deadline e_j of arrival), distance (i.e., the travelling distance is below d_i), and budget (i.e., the travelling cost is below task budget B_j). We say that the worker-and-task assignment pair $\langle w_i, t_j \rangle$ is *valid*, if there is an edge between vertices w_i and t_j in the graph.

As an example in Figure 5(a), we have a worker set $W_p = \{w_i | 1 \leq i \leq 5\}$, and a spatial task set $T_p = \{t_j | 1 \leq j \leq 3\}$, which are denoted by two types of vertices (i.e., represented by rectangle and circle shapes, respectively) in a bipartite graph. Any edge connects two types of vertices w_i and t_j, if worker w_i can reach the location of task t_j and do tasks with the required skills from t_j. For example, there exists an edge between w_1 and t_1, which indicates that worker w_1 can move to the location of t_1 before the arrival deadline e_1, with the travelling distance under d_1, with the travelling cost below budget B_1, and moreover with some skill(s) in the required skill set Y_1 of task t_1.



(a) Original MS-SC Problem (b) Decomposed Subproblems

Fig. 5: Illustration of Decomposing the MS-SC Problem.

Note that, one or multiple worker vertices (e.g., w_1, w_3, and w_4) may be connected to the same task vertex (e.g., t_1). Furthermore, multiple task vertices, say t_1 and t_2, may also share some conflicting workers (e.g., w_3 or w_4), where the conflicting worker w_3 (or w_4) can be assigned to either task t_1 or task t_2 mutual exclusively.

Procedure MS-SC_Decomposition {
Input: n workers in W_p, m time-constrained spatial tasks in T_p, and the number of groups g
Output: decomposed MS-SC subproblems, P_s $(1 \leq s \leq g)$
(1) for $s = 1$ to g
(2) $P_s = \emptyset$
(3) compute all valid worker-and-task pairs $\langle w_i, t_j \rangle$ from W_p and T_p, and obtain a bipartite graph G
(4) for $s = 1$ to g
(5) let set $T_p^{(j)}$ contain the next anchor task t_j and its top-$(\lceil m/g \rceil - 1)$ nearest tasks // the task, t_j, whose longitude is the smallest
(6) for each task vertex $t_j \in T_p^{(j)}$ in graph G
(7) obtain all worker vertices w_i that connect with task vertex t_j
(8) add all pairs $\langle w_i, t_j \rangle$ to P_s
(9) return P_s (for $1 \leq s \leq g$)
}

Fig. 6: The MS-SC Problem Decomposition Algorithm.

Decomposing the MS-SC Problem. Next, we will illustrate how to decompose the MS-SC problem, with respect to task vertices in the bipartite graph. Figure 5 shows an example of decomposing the MS-SC problem (as shown in Figure 5(a)) into 3 subproblems (as depicted in Figure 5(b)), where each subproblem contains a subgroup of one single spatial task (i.e., group size = 1), associated with its connected worker vertices. For example, the first subgroup in Figure 5(b)) contains task vertex t_1, as well as its connecting worker vertices w_1, w_3, and w_4. Different task vertices may have conflicting workers, for example, tasks t_1 and t_2 share the same (conflicting) worker vertices w_3 and w_4.

In a general case, given n workers and m spatial tasks, we partition the bipartite graph into g subgroups, each of which contains $\lceil m/g \rceil$ spatial tasks, as well as their connecting workers. Figure 6 presents the pseudo code of our MS-SC problem decomposition algorithm, namely MS-SC_Decomposition, which returns g MS-SC subproblems (each corresponding to a subgroup with $\lceil m/g \rceil$ tasks), P_s, after decomposing the original MS-SC problem.

Specifically, we first initialize g empty subproblems, P_s, where $1 \leq s \leq g$ (lines 1-2). Then, we find out all valid worker-and-task pairs $\langle w_i, t_j \rangle$ in the crowdsourcing system at timestamp p, which can form a bipartite graph G, where valid pairs satisfy the constraints of skills, times, distances, and budgets (line 3).

Next, we want to obtain one subproblem P_s at a time (lines 4-8). In particular, for each round, we retrieve an anchor task t_j and its top-$(\lceil m/g \rceil - 1)$ nearest tasks, which form a task set $T_p^{(j)}$ of size $\lceil m/g \rceil$ (line 5). Here, we choose anchor tasks with a sweeping style, that is, we always choose the task whose longitude is smallest (in the case where multiple tasks have the same longitude, we choose the one with smallest latitude). Then, for each task $t_j \in T_p^{(j)}$, we obtain its corresponding vertex in G and all of its connecting worker vertices w_i, and add pairs $\langle w_i, t_j \rangle$ to subproblem P_s (lines 6-8). Finally, we return all the g decomposed subproblems P_s.

5.2 The g-D&C Algorithm

In this subsection, we propose an efficient *g-divide-and-conquer* (g-D&C) algorithm, namely MS-SC_gD&C, which recursively partitions the original MS-SC problem into subproblems, solves each subproblem (via recursion), and merges assignment results of subproblems by resolving the conflicts.

Specifically, in Algorithm MS-SC_gD&C, we first estimate the best number of groups, g, to partition, with respect to W_p and T_p, which is based on the cost model proposed later in Section 5.4 (line 2). Then, we will call the MS-SC_Decomposition algorithm (as mentioned in Figure 6) to obtain subproblems P_s (line 3). For each subproblem P_s, if P_s involves more than 1 task, then we can recursively call Algorithm MS-SC_gD&C itself, by further dividing the subproblem P_s (lines 5-6). Otherwise, when subproblem P_s contains only one single task, we apply the greedy algorithm of the classical set cover problem for task set $T_p(P_s)$ and worker set $W_p(P_s)$ (lines 7-8).

After that, we can obtain an assignment instance set $I_p^{(s)}$ for each subproblem P_s, and merge them into one single worker-and-task assignment instance set I_p, by reconciling the conflict (lines 9-11). In particular, I_p is initially empty (line 1), and

each time merged with an assignment set $I_p^{(s)}$ from subproblem P_s (lines 10-11). Due to the confliction among subproblems, we call function MS-SC_Conflict_Reconcile $(\cdot, \cdot)$ (discussed later in Section 5.3) to resolve the confliction issue during the merging process. Finally, we can return the merged assignment instance set I_p (line 12).

Procedure MS-SC_gD&C {
Input: n workers in W_p, and m time-constrained spatial tasks in T_p
Output: a worker-and-task assignment instance set, I_p
(1) $I_p = \emptyset$
(2) estimate the best number of groups, g, for W_p and T_p
(3) invoke MS-SC_Decomposition(W_p, T_p, g), and obtain subproblems P_s
(4) for $s = 1$ to g
(5) if the number of tasks in subproblem P_s (group size) is greater than 1
(6) $I_p^{(s)}$ = MS-SC_gD&C$(W_p(P_s), T_p(P_s))$
(7) else
(8) invoke classical greedy set cover algorithm to solve subproblem P_s, and obtain assignment results $I_p^{(s)}$
(9) for $i = 1$ to g
(10) find the next subproblem, P_s
(11) I_p = MS-SC_Conflict_Reconcile $(I_p, I_p^{(s)})$
(12) return I_p
}

Fig. 7: The g-Divide-and-Conquer Algorithm.

5.3 Merging Conflict Reconciliation

In this subsection, we introduce the merging conflict reconciliation procedure, which resolves the conflicts while merging assignment results of subproblems (i.e., line 11 of Procedure MS-SC_gD&C). Assume that I_p is the current assignment instance set we have merged so far. Given a new subproblem P_s with assignment set $I_p^{(s)}$, Figure 8 shows the merging algorithm, namely MS-SC_Conflict_Reconcile, which combines two assignment sets I_p and $I_p^{(s)}$ by resolving conflicts.

Procedure MS-SC_Conflict_Reconcile {
Input: the current assignment instance set, I_p, of subproblem P we have merged, and the assignment instance set, $I_p^{(s)}$, of subproblem P_s
Output: a merged worker-and-task assignment instance set, I_p
(1) let W_c be a set of all conflicting workers between I_p and $I_p^{(s)}$
(2) while $W_c \neq \emptyset$
(3) choose a worker $w_i \in W_c$ with the highest travelling cost in $I_p^{(s)}$
(4) if we substitute w_i with w_i' in P_s having the highest score $S_p^{(s)}$
(5) compute the reduction of the assignment score, $\Delta S_p^{(s)}$
(6) if we substitute w_i with w_i'' in P having the highest score S_p
(7) compute the reduction of the assignment score, ΔS_p
(8) if $\Delta S_p > \Delta S_p^{(s)}$
(9) substitute worker w_i with w_i' in $I_p^{(s)}$
(10) else
(11) substitute worker w_i with w_i'' in I_p
(12) $W_c = W_c - \{w_i\}$
(13) $I_p = I_p \cup I_p^{(s)}$
(14) return I_p
}

Fig. 8: The Merging Conflict Reconciliation Algorithm.

In particular, two distinct tasks from two subproblems may be assigned with the same (conflicting) worker w_i. Since each worker can only be assigned to one spatial task at a time, we thus need to avoid such a scenario when merging assignment instance sets of two subproblems (e.g., I_p and $I_p^{(s)}$). Our algorithm in Figure 8 first obtain a set, W_c, of all conflicting workers between I_p and $I_p^{(s)}$ (line 1). Then, each time we greedily solve the conflicts for workers w_i in an non-decreasing order of the travelling cost (i.e., c_{ij}) in $I_p^{(s)}$ (line 3). Next, in order to resolve the conflicts, we try to replace worker w_i with another worker w_i' (or w_i'') in P_s (or P) with the highest score $S_p^{(s)}$ (or S_p), and compute possible reduction of the assignment score, $\Delta S_p^{(s)}$ (or ΔS_p) (lines 4-7). Note that, here we replace worker w_i with other available workers. If no other workers are available for replacing w_i, we may need to sacrifice task t_j that worker w_i is assigned to. For example, when we cannot find another worker to replace w_i in P_s, the substitute of w_i will be set as an empty worker, which means the assigned task t_j for w_i in $I_p^{(s)}$ will be sacrificed and $\Delta S_p^{(s)} = B_j'$ (as calculated in Equation 2). In the case that $\Delta S_p > \Delta S_p^{(s)}$, we substitute worker w_i with w_i' in $I_p^{(s)}$ (since the replacement of w_i in subproblem $S_p^{(s)}$ leads to lower score reduction); otherwise, we resolve conflicts by replacing w_i with w_i'' in I_p (lines 8-12). After resolving all conflicts, we merge assignment instance set I_p with $I_p^{(s)}$ (line 13), and return the merged result I_p.

5.4 Cost-Model-Based Estimation of the Best Number of Groups

In this subsection, we discuss how to estimate the best number of groups, g, such that the total cost of solving the MS-SC problem in g-divide-and-conquer approach is minimized. Specifically, the cost of the g-divide-and-conquer approach consists of 3 parts: the cost, F_D, of decomposing subproblems, that, F_C, of conquering subproblems recursively, and that, F_M, of merging subproblems by resolving conflicts.

Without loss of generality, as illustrated in Figure 9, during the g-divide-and-conquer process, on level k, we recursively divide the original MS-SC problem into g^k subproblems, $P_1^{(k)}$, $P_2^{(k)}$, ..., and $P_{g^k}^{(k)}$, where each subproblem involves m/g^k spatial tasks.

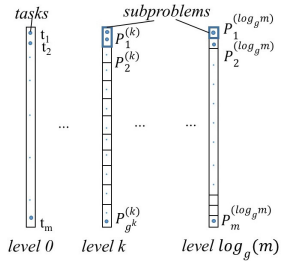


Fig. 9: Illustration of the Cost Model Estimation.

The Cost, F_D, of Decomposing Subproblems. From Algorithm MS-SC_Decomposition (in Figure 6), we first need to retrieve all valid worker-and-task assignment pairs (line 3), whose cost is $O(m \cdot n)$. Then, we will divide each problem into g subproblems, whose cost is given by $O(m \cdot g + m)$ on each level. For level k, we have m/g^k tasks in each subproblem $P_i^{(k)}$. We will further divide it into g more subproblems, $P_j^{(k+1)}$, and each one will have m/g^{k+1} tasks. To obtain m/g^{k+1} tasks in each subproblem $P_j^{(k+1)}$, we first need to find the anchor task, which needs $O(m/g^k)$ cost, and further retrieve the rest tasks, which needs $O(m/g^{k+1})$ cost. Moreover, since we will have g^{k+1} subproblems on level $k+1$, the cost of decomposing tasks on level k is given by $O(m \cdot g + m)$.

Since there are totally $log_g(m)$ levels, the total cost of decomposing the MS-SC problem is given by:

$$F_D = m \cdot n + (m \cdot g + m) \cdot log_g(m).$$

The Cost, F_C, of Recursively Conquering Subproblems. Let function $F_C(x)$ be the total cost of conquering a subproblem which contains x spatial tasks. Then, we have the following recursive function:

$$F_C(m) = g \cdot F_C(\left\lceil \frac{m}{g} \right\rceil).$$

Assume that deg_t is the average degree of task nodes in the bipartite group G. Then, the base case of function $F_C(x)$ is the case that $x = 1$, in which we apply the greedy algorithm on just one single task and deg_t workers. Thus, by the analysis of the time complexity in Section 4.3, we have:

$$F_C(1) = cost_{greedy}(deg_t, 1) = deg_t^2.$$

From the recursive function $F_C(x)$ and its base case, we can obtain the total cost of the recursive invocation on levels from 1 to $log_g(m)$ below:

$$\sum_{k=1}^{log_g(m)} F_c(m/g^k) = \frac{1-m}{1-g} deg_t^2$$

The Cost, F_M, of Merging Subproblems. Next, we provide the cost, F_M, of merging subproblems by resolving conflicts. Assume that we have n_s workers who could be assigned to more than one spatial task (i.e., conflicting workers). Moreover, each worker node has an average degree deg_w in the bipartite graph. During the subproblem merging processing, we can estimate the worst-case cost of resolving conflicts for these n_s workers, and we may resolve conflicts for each worker at most $(deg_w - 1)$ times.

Therefore, the worst-case cost of merging subproblems can be given by: $F_M = n_s \cdot (deg_w - 1)$.

The Total Cost of the g-D&C Approach. The total cost, $cost_{gD\&C}$, of the g-D&C algorithm can be given by summing up three costs, F_D, F_C, and F_M. That is, we have:

$$\begin{aligned} cost_{gD\&C} &= F_D + \sum_{k=1}^{log_g(m)} F_c(m/g^k) + F_M \quad (4) \\ &= (mg + m)\log_g(m) + \frac{1-m}{1-g} deg_t^2 + n_s(deg_w - 1). \end{aligned}$$

We take the derivation of $cost_{gD\&C}$ (given in Eq. (4)) over g, and let it be 0. In particular, we have:

$$\frac{\partial cost_{gD\&C}}{\partial g} = \frac{m\log(m)(g\log(g) - g - 1)}{g\log(2g)} + \frac{1-m}{(1-g)^2} deg_t^2 = 0 \quad (5)$$

We notice that when $g = 2$, $\frac{\partial cost_{gD\&C}}{\partial g}$ is much smaller than 0 but increases quickly when g grows. In addition, g can only be an integer. Then we can try the integers, (2, 3, 4...), until $\frac{\partial cost_{gD\&C}}{\partial g}$ is above 0.

6 The Cost-Model-Based Adaptive Algorithm

In this section, we introduce a *cost-model-based adaptive approach*, which adaptively decides the strategies to apply our proposed greedy and g-divide-and-conquer (g-D&C) algorithms. The basic idea is as follows. Unlike the g-D&C algorithm, we do not divide the MS-SC problem into subproblems recursively until task group sizes become 1 (which can be solved by the greedy algorithm of set cover problems). Instead, based on our proposed cost model, we will partition the problem into subproblems, and adaptively determine when to stop in some partitioning round (i.e., the total cost of solving subproblems with the greedy algorithm is smaller than that of continuing dividing subproblems).

Procedure MS-SC_Adaptive {
Input: n workers in W_p, and m time-constrained spatial tasks in T_p
Output: a worker-and-task assignment instance set, I_p
(1) $I_p = \emptyset$
(2) estimate the cost, $cost_{greedy}$, of the greedy algorithm
(3) estimate the best number of groups, g, and obtain the cost, $cost_{gdc}$, of the g-D&C approach
(4) if $cost_{greedy} < cost_{gdc}$
(5) I_p = MS-SC_Greedy(W_p, T_p)
(6) else // *g-D&C algorithm*
(7) invoke MS-SC_Decomposition(W_p, T_p, g), and obtain subproblems P_s
(8) for each subproblem, P_s,
(9) $I_p^{(s)}$ = MS-SC_Adaptive($W_p(P_s)$, $T_p(P_s)$)
(10) for $i = 1$ to g
(11) find the next subproblem, P_s
(12) I_p = MS-SC_Conflict_Reconcile (I_p, $I_p^{(s)}$)
(13) return I_p
}

Fig. 10: The MS-SC Cost-Model-Based Adaptive Algorithm.

6.1 Algorithm of the Cost-Model-Based Adaptive Approach

Figure 10 shows the pseudo-code of our cost-model-based adaptive algorithm, namely MS-SC_Adaptive. Initially, we estimate the cost, $cost_{greedy}$, of applying the greedy approach over worker/task sets W_p and T_p (line 2). Similarly, we also estimate the best group size, g, and compute the cost, $cost_{gd\&c}$ of using the g-D&C algorithm (line 3). If it holds that the cost of the greedy algorithm is smaller than that of the g-D&C approach (i.e., $cost_{greedy} < cost_{gdc}$), then we will use the greedy algorithm by invoking function MS-SC_Greedy$(\cdot, \cdot)$ (due to its lower cost; lines 4-5). Otherwise, we will apply the g-D&C algorithm, and further partition the problem into subproblems P_s (lines 6-7). Then, for each subproblem P_s, we recursively call the cost-model-based adaptive algorithm, and retrieve the assignment instance set $I_p^{(s)}$ (line 9). After that, we merge all the assignment instance sets from subproblems by invoking function MS-SC_Conflict_Reconcile$(\cdot, \cdot)$ (lines 10-12). Finally, we return the worker-and-task assignment instance set I_p (line 13).

6.2 Cost Model for the Stopping Condition

Next, we discuss how to determine the stopping level, when using our cost-model-based adaptive approach to recursively solve the MS-SC problem. Intuitively, at the current level k, we need to estimate the costs, $cost_{greedy}$ and $cost_{gdc}$, of using greedy and g-D&C algorithms, respectively, to solve the remaining MS-SC problem. If the greedy algorithm has lower cost, then we will stop the divide-and-conquer, and apply the greedy algorithm for each subproblems.

In the sequel, we discuss how to obtain the formulae of costs $cost_{greedy}$ and $cost_{gdc}$.

The Cost, $cost_{greedy}$, of the Greedy Algorithm. Given a set, W_p, of n workers and a set, T_p, of m tasks, the cost, $cost_{greedy}$, of our greedy approach (as given in Figure 3) has been discussed in Section 4.3.

In the bipartite graph of valid worker-and-task pairs, denote the average degree of workers as deg_w, and that of tasks as deg_t. In Figure 3, the computation of valid worker-and-task pairs in line 2 needs $O(m \cdot n)$ cost. Since there are at most n

iterations, for each round (lines 3-16), we apply two worker-pruning methods to at most $(2m \cdot deg_t)$ pairs, and select pairs with the highest score increases, which need $O(3m \cdot n \cdot deg_t)$ cost in total. For the cost of task-pruning, there are totally n rounds (lines 3-16; i.e., removing one out of n workers in each round in line 16). In each round, there are at most deg_w out of m tasks (line 5) that may be potentially pruned by Lemma 4 (line10). To check each of deg_w tasks, we need $O(deg_t)$ cost. Therefore, the total cost of task-pruning is given by $O(n \cdot deg_t \cdot deg_w)$. If we cannot prune a task that was assigned with a worker in the last round (lines 3-16), then we need to update score increases of deg_t workers for that task. Each task will be assigned with workers for deg_t times. Thus, the total update cost for one task is given by $O(deg_t^2)$ (line 12). Therefore, $cost_{greedy}(n, m)$ can be given by:

$$\begin{aligned} & cost_{greedy}(n, m) \\ = \quad & C_{greedy} \cdot (m \cdot n + n \cdot deg_t \cdot (3m + deg_w) + m \cdot deg_t^2), \end{aligned} \tag{6}$$

where parameter C_{greedy} is a constant factor, which can be inferred from cost statistics of the greedy algorithm.

The Cost, $cost_{gdc}$, of the g-D&C Algorithm. Assume that the current g-divide-and-conquer level is k. We can modify the cost analysis in Section 5.4, by considering the cost, $cost_{gdc}$, of the remaining divide-and-conquer levels. Specifically, we have the cost, F'_D, of the decomposition algorithm, that is:

$$F'_D = m \cdot n + (m \cdot g + m) \cdot k.$$

Moreover, when the current level is k, the cost of conquering the remaining subproblems is given by:

$$\sum_{i=k}^{log_g(m)} F_c(m/g^i).$$

Finally, the cost of merging subproblems is given by F_M.

As a result, the total cost, $cost_{gdc}$, of solving the MS-SC problem with our g-D&C approach for the remaining partitioning levels (from level k to $log_g(m)$) can be given by:

$$cost_{gdc} = C_{gdc} \cdot (F'_D + \sum_{i=k}^{log_g(m)} F_c(m/g^i) + F_M),$$

where parameter C_{gdc} is a constant factor, which can be inferred from time cost statistics of the g-D&C algorithm.

This way, we compare $cost_{greedy}$ with $cost_{gdc}$ (as mentioned in line 4 of MS-SC_Adaptive Algorithm). If $cost_{greedy}$ is smaller than $cost_{gdc}$, we stop at the current level k, and apply the greedy algorithm to tackle the MS-SC problem directly; otherwise, we keep dividing the original MS-SC problem into subproblems (i.e., g-D&C).

Discussions on 3 MS-SC Approaches. The greedy approach (GREEDY) greedily assigns workers to tasks to maximize the increase of the assignment score in each iteration, which may achieve a local optimality of the whole problem space. The *g-divide-and-conquer* approach (g-D&C) keeps dividing the original problem into g smaller subproblems on each level, until the number of tasks in each subproblem is 1. For each one-task subproblem, we use the state-of-the-art *set cover greedy algorithm* (SCGreedy) [22], a $\ln(N)$-approximation algorithm, to solve it. For each task, the sum of the travelling costs calculated by GREEDY cannot be less than that calculated by SCGreedy such that the score achieved by GREEDY is less than that achieved by SCGreedy. The reason is that, for each task, GREEDY cannot guarantee the same "best" worker as that selected by SCGreedy, since the "best" worker may have been assigned to some other task. In other words, g-D&C can achieve better local optimal results for one-task subproblems compared with GREEDY. In addition, when we combine the results of subproblems, we solve the conflicts and maintain these better local optimal results. As a result, g-D&C can achieve better assignment scores. The *adaptive cost-model-based* approach trades the accuracy for the running time such that it can run faster than g-D&C and achieve better results than GREEDY.

7 Experimental Study

7.1 Experimental Methodology

Data Sets. We use both real and synthetic data to test our proposed MS-SC approaches. Specifically, for real data, we use Meetup data set from [19], which was crawled from *meetup.com* between Oct. 2011 and Jan. 2012. There are 5,153,886 users, 5,183,840 events, and 97,587 groups in Meetup, where each user is associated with a location and a set of tags, each group is associated with a set of tags, and each event is associated with a location and a group who created the event. For an event, we use the tags of the group who creates the event as its tags. To conduct the experiments on our approaches, we use the locations and tags of users in Meetup to initialize the locations and the practiced skills of workers in our MS-SC problem. In addition, we utilize the locations and tags of events to initialize the locations and the required skills of tasks in our experiments. Since workers are unlikely to move between two distant cities to conduct one spatial task, and the constraints of time (i.e., e_j), budget (i.e., B_j) and distance (i.e., d_i) also prevent workers from moving too far, we only consider those user-and-event pairs located in the same city. Specifically, we select one famous and popular city, Hong Kong, and extract Meetup records from the area of Hong Kong (with latitude from $22.209°$ to $113.843°$ and longitude from $22.609°$ to $114.283°$), in which we obtain 1,282 tasks and 3,525 workers.

For synthetic data, we generate locations of workers and tasks in a 2D data space $[0, 1]^2$, following either Uniform (UNIFORM) or Skewed (SKEWED) distribution. For Uniform distribution, we uniformly generate the locations of tasks/workers in the 2D data space. Similarly, we also generate tasks/workers with the Skewed distribution by locating 90% of them into a Gaussian cluster (centered at (0.5, 0.5) with variance = 0.2^2), and distribute the rest workers/tasks uniformly. Then, for skills of each worker, we randomly associate one user in Meetup data set to this worker, and use tags of the user as his/her skills in our MS-SC system. For the required skills of each task, we randomly select an event, and use its tags as the required skills of the task.

For both real and synthetic data sets, we simulate the velocity of each worker with Gaussian distribution within range $[v^-, v^+]$, for $v^-, v^+ \in (0, 1)$. For the unit price, C_i, w.r.t. the travelling distance of each worker, we generate it following the Uniform distribution within the range $[C^-, C^+]$. Furthermore, we produce the maximum moving distance of

TABLE 4: Experiments Settings.

Parameters	Values
the number of tasks m	1K, 2K, **5K**, 8K, 10K
the number of workers n	1K, 2K, **5K**, 8K, 10K
the task budget range $[B^-, B^+]$	[1, 5], **[5, 10]**, [10, 15], [15, 20], [20, 25]
the velocity range $[v^-, v^+]$	[0.1, 0.2], **[0.2, 0.3]**, [0.3, 0.4], [0.4, 0.5]
the unit price w.r.t. distance $[C^-, C^+]$	[10, 20], **[20, 30]**, [30, 40], [40, 50]
the moving distance range $[d^-, d^+]$	[0.1, 0.2], [0.2, 0.3], **[0.3, 0.4]**, [0.4, 0.5]
the expiration time range $[rt^-, rt^+]$	[0.25, 0.5], [0.5, 1], **[1, 2]**, [2, 3], [3, 4]

each worker, following the Uniform distribution within the range $[d^-, d^+]$ (for $d^-, d^+ \in (0, 1)$). For temporal constraints of tasks, we also generate the arrival deadlines of tasks, e, within range $[rt^-, rt^+]$ with Gaussian distribution. Finally, we set the budgets of tasks with Gaussian distribution within the range $[B^-, B^+]$. Here, for Gaussian distributions, we linearly map data samples within $[-1, 1]$ of a Gaussian distribution $\mathcal{N}(0, 0.2^2)$ to the target ranges.

MS-SC Approaches and Measures. We conduct experiments to compare our three approaches, GREEDY, g-D&C and ADAPTIVE, with a random method, namely RANDOM, which randomly assigns workers to tasks.

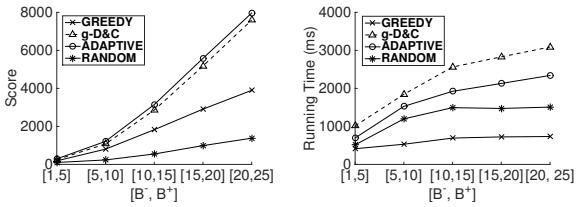
In particular, GREEDY selects a "best" worker-and-task assignment with the highest score increase each time, which is a local optimal approach. The g-D&C algorithm keeps dividing the problem into g subproblems on each level, until finally the number of tasks in each subproblem is 1 (which can be solved by the greedy algorithm on each one-task subproblem). Here, the parameter g can be estimated by a cost model to minimize the computing cost. The cost-model-base adaptive algorithm (ADAPTIVE) makes the trade-off between GREEDY and g-D&C, in terms of efficiency and accuracy, which adaptively decides the stopping level of the divide-and-conquer. To evaluate our three proposed approaches, we need to compare the results with ground truth. However, as proved in Section 2.4, the MS-SC problem is NP-hard, and thus infeasible to calculate the real optimal result as the ground truth. Alternatively, we will compare the effectiveness of our three approaches with that of a random (RANDOM) method, which randomly chooses a worker, and then randomly assigns him/her to a task that he/she can satisfy its constraints and the required skills. For each MS-SC instance, we run RANDOM for 10 times, and report the highest score. We also conducted comparison experiments on a small dataset to show that the results achieved by our three approaches are close to the optimal results. Due to the space limitation, please refer to Appendix H of supplementary materials.

Table 4 depicts our experimental settings, where the default values of parameters are in bold font. In each set of experiments, we vary one parameter, while setting other parameters to their default values. For each experiment, we report the running time and the assignment score of our tested approaches. The trend w.r.t. the number of the completed tasks is similar to that of the assignment score. Due to space limitations, please refer to experimental results for the number of the completed tasks in Appendix I of supplementary materials. All our experiments were run on an Intel Xeon X5675 CPU @3.07 GHZ with 32 GB RAM in Java.

7.2 Experiments on Real Data

In this subsection, we show the effects of the range of task budgets $[B^-, B^+]$, the range of workers' velocities $[v^-, v^+]$, and the range of unit prices w.r.t. distance $[C^-, C^+]$.

Effect of the Range of Task Budgets $[B^-, B^+]$. Figure 11 illustrates the experimental results on different ranges, $[B^-, B^+]$, of task budgets B_j from $[1, 5]$ to $[20, 25]$. In Figure 11(a), the assignment scores of all the four approaches increase, when the value range of task budgets gets larger. When the average budgets of tasks increase, the flexible budget B' of each task will also increase. g-D&C and ADAPTIVE can achieve higher score than GREEDY. In contrast, RANDOM has the lowest score, which shows that our proposed three approaches are more effective. As shown in Figure 11(b), the running times of our three approaches increase, when the range of task budgets becomes larger. The reason is that, when $B_j \in [B^-, B^+]$ increases, each task has more valid workers, which thus leads to higher complexity of the MS-SC problem and the increase of the running time. RANDOM is the fastest (however, with the lowest assignment score), since it does not even need to find local optimal assignment. ADAPTIVE achieves much lower running time than g-D&C (a bit higher time cost than GREEDY), but has comparable score with g-D&C (much higher score than GREEDY), which shows the good performance of ADAPTIVE, compared with GREEDY and g-D&C. The requesters provide budgets to support the travelling costs of workers. Higher travelling budget can support workers located in farther locations, which means more worker candidates that can reach the task.



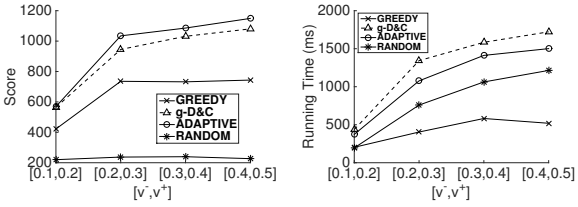
(a) Scores of Assignment (b) Running Times

Fig. 11: Effect of the Range of Task Budgets $[B^-, B^+]$ (Real Data).

Effect of the Workers' Velocity Range $[v^-, v^+]$. Figure 12 reports the effect of the range of velocities, $[v^-, v^+]$, of workers over real data. As shown in Figure 12(a), when the range of velocities increases from $[0.1, 0.2]$ to $[0.2, 0.3]$, the scores of all the approaches first increase; then, they stop growing for the velocity range varying from [0.2, 0.3] to [0.4, 0.5]. The reason is that, at the beginning, with the increase of velocities, workers can reach more tasks before their arrival deadlines. Nevertheless, workers are also constrained by their maximum moving distances, which prevents them from reaching more tasks. ADAPTIVE can achieve a bit higher scores than g-D&C, and much better assignment scores than GREEDY.

In Figure 12(b), when the range of velocity $[v^-, v^+]$ increases, the running times of our tested approaches also increase, due to the cost of more valid worker-and-task pairs to be handled. For RANDOM, in each iteration, it may need more time to eliminate the invalid pairs caused by the newly assigned worker-and-task pairs, which leads to the increase of

the total running time. Similar to previous results, RANDOM is the fastest, and g-D&C is the slowest. ADAPTIVE requires about 0.5-1.5 seconds, and has lower time cost than g-D&C, which shows the efficiency of our proposed approaches.

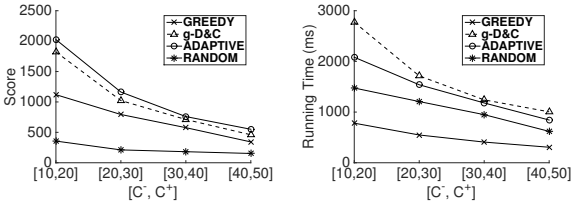


(a) Scores of Assignment (b) Running Times

Fig. 12: Effect of the Range of Velocities $[v^-, v^+]$ (Real Data).

Effect of the Range of Unit Prices w.r.t. Travelling Distance $[C^-, C^+]$. In Figure 13(a), when the unit prices w.r.t. travelling distance $C_i \in [C^-, C^+]$ increase, the scores of all the approaches decrease. The reason is that, when the range of unit prices $[C^-, C^+]$ increases, we need to pay more wages containing the travelling costs of workers (to do spatial tasks), which in turn decreases the flexible budget of each task. However, ADAPTIVE can still achieve the highest score among all four approaches; scores of g-D&C are close to the scores of ADAPTIVE and higher than GREEDY. This parameter can be determined by

In Figure 13(b), when the range of unit prices, $[C^-, C^+]$, of the travelling cost increases, the number of valid worker-and-task pairs decreases, and thus the running time of all the approaches also decreases. Our ADAPTIVE algorithm is faster than g-D&C, and slower than GREEDY. On a real platform, the unit price value can be set by the platform based on the gas price and the gas consumption per mile for specific vehicles.



(a) Scores of Assignment (b) Running Times

Fig. 13: Effect of the Range of Unit Prices w.r.t. Travelling Distance $[C^-, C^+]$ (Real Data).

In addition, we also tested the effects of the range, $[d^-, d^+]$, of maximum moving distances for workers, and the expiration time range, $[rt^-, rt^+]$, of tasks over the real data set, Meetup. Due to space limitations, please refer to experimental results with respect to other parameters (e.g., $[d^-, d^+]$ and $[rt^-, rt^+]$) in Appendix F of supplementary materials.

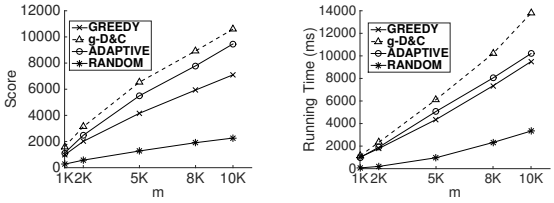
From experimental results on the real data above, ADAPTIVE can achieve higher scores than Greedy and g-D&C, and it is faster than g-D&C and slower than GREEDY. Although g-D&C can achieve good scores close to ADAPTIVE, it is the slowest among all the 4 approaches.

7.3 Experiments on Synthetic Data

In this subsection, we test the effectiveness and robustness of our three MS-SC approaches, GREEDY, g-D&C, and ADAPTIVE, compared with RANDOM, by varying the number of tasks m and the number of workers n on synthetic data sets. Due to space limitations, we present the experimental results for tasks/workers with Uniform distributions. For similar results with tasks/workers following skewed distributions, please refer to Appendix G in supplementary materials.

Effect of the Number of Tasks m. Figure 14 illustrates the effect of the number, m, of spatial tasks, by varying m from $1K$ to $10K$, over synthetic data sets, where other parameters are set to default values. For assignment scores in Figure 14(a), g-D&C obtains results with the highest scores among all the four approaches. ADAPTIVE performs similar to g-D&C, and achieves good results similar to g-D&C. GREEDY is not as good as g-D&C and ADAPTIVE, but is still much better than RANDOM. When the number, m, of spatial tasks becomes larger, all our approaches can achieve higher scores.

In Figure 14(b), when m increases, the running time also increases. This is because, we need to deal with more worker-and-task assignment pairs for large m. The ADAPTIVE algorithm is slower than GREEDY, and faster than g-D&C. In addition, we find that the running time of GREEDY performs, with the same trend as that estimated in our cost model (as given in Eq. (6)).



(a) Scores of Assignment (b) Running Times

Fig. 14: Effect of the Number of Tasks m (Synthetic Data).

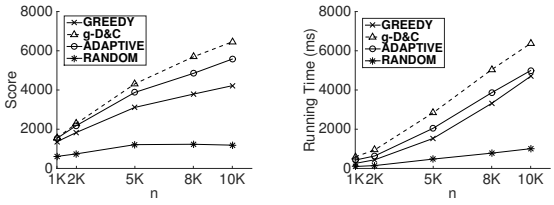
Effect of the Number of Workers n. Figure 15 shows the experimental results with different numbers of workers, n, from $1K$ to $10K$ over synthetic data, where other parameters are set to their default values. Similar to previous results about the effect of m, in Figure 15(a), our proposed three approaches can obtain good results with high assignment scores, compared with RANDOM. Moreover, when the number, n, of workers increases, the scores of all our approaches also increase. The reason is that, when n increases, we have more potential workers, who can be assigned to nearby tasks, which may lead to even larger scores.

In Figure 15(b), the running time of our approaches increases, with the increase of the number of workers . This is due to higher cost to process more workers (i.e., larger n). Similarly, ADAPTIVE has higher time cost than GREEDY, and lower time cost than g-D&C.

In summary, over synthetic data sets, our ADAPTIVE algorithm trades the accuracy for efficiency, and thus has the trade-off of scores/times between GREEDY and g-D&C.

8 Related Work

In this section, we review the related work on spatial crowdsourcing, as well as the set cover problem.



(a) Scores of Assignment (b) Running Times

Fig. 15: Effect of the Number of Workers n (Synthetic Data).

Spatial Crowdsourcing. Without considering the location information in crowdsourcing, previous works [7], [25], [13] studied the task assignment to achieve better accuracy, and prior works [8], [24] studied how to select a proper worker set for a particular task. Prior works like [5], [14] usually studied crowdsourcing problems, which treat the location information as a parameter and distribute tasks to workers. In these problems, workers are not required to accomplish tasks on sites. In our MS-SC problem, we focus on finding an assignment such that the spatial (e.g., maximum moving distances of worker) and temporal (e.g., the arrival deadlines of tasks) constraints can be met, the skills required by the tasks can be supported by workers, and the assignment score is maximized. Thus the existing methods cannot be directly applied.

The spatial crowdsourcing platform [17] requires workers to physically move to some specific locations, and perform the requested services, such as taking photos/videos, waiting in line at shopping malls, and decorating a room. As an example, some previous works [11], [15] studied the small-scale or specified campaigns benefiting from *participatory sensing* techniques, which utilize smart devices (equipped by workers) to sense/collect data for real applications.

Kazemi and Shahabi [17] classified the spatial crowdsourcing systems from two perspectives: people's motivation and publishing models. From the perspective of people's motivation, the spatial crowdsourcing can be categorized into two groups: reward-based, in which workers can receive rewards according to the services they supplied, and self-incentivised, in which workers conduct tasks voluntarily. In our work, we study our MS-SC problem based on the reward-based model, where workers are paid for doing tasks. However, with a different goal, our MS-SC problem targets at assigning workers to tasks by using our proposed algorithms, such that the required skills of tasks can be covered, and the total reward budget (i.e., flexible budget B'_j in Eq. (2)) can be maximized. Note that, we can embed incentive mechanisms from existing works [20], [23] into our MS-SC framework to distribute rewards (flexible budgets) among workers, which is however not the focus of our problem.

According to the publishing modes of spatial tasks, the spatial crowdsourcing can be also classified into two categories: *worker selected tasks* (WST) and *server assigned tasks* (SAT) [17]. In particular, for the WST mode, spatial tasks are broadcast to all workers, and workers can select any tasks by themselves. In contrast, for the SAT mode, the spatial crowdsourcing server will directly assign tasks to workers, based on location information of tasks/workers.

Some prior works [5], [12] on the WST mode allowed workers to select available tasks, based on their personal preferences. However, for the SAT mode, previous works focused on assigning available workers to tasks in the system, such that the number of assigned tasks on the server side [17], the number of worker's self-selected tasks on the client side [12], or the reliability-and-diversity score of assignments [10] is maximized.

In particular, Cheng et al. [10] tackles the problem of reliable diversity-based spatial crowdsourcing (RDB-SC), which finds a worker-and-task assignment strategy that maximizes the assignment score (w.r.t. spatial/temporal diversity and reliability of tasks). In contrast, our MS-SC problem has a different, yet more general, goal, which involves multi-skilled workers and complex tasks with the required skills (not studied before), and aims to maximize a different assignment score (i.e., flexible budget, given by the total budget of the completed tasks minus the total travelling cost of workers). In addition, our MS-SC problem also needs to consider several constraints, such as skill-covering, budget, time, and distance constraints, which make our problem more challenging.

Due to different assignment goals, for example, between RDB-SC [10] and MS-SC, we cannot directly borrow previous techniques such as [10], [12], [17] to tackle the MS-SC problem. For instance, the greedy algorithm should design effective approach to find one assignment each time with the highest increase of flexible budget in our MS-SC problem (rather than highest reliability and diversity as discussed in RDB-SC [10]); for g-D&C, we propose an effective cost model to determine the best g value to maximize the MS-SC performance (instead of always dividing the problem into 2 subproblems in [10]); most importantly, we also propose a novel cost-model-based adaptive algorithm, which combines the greedy and g-D&C algorithms based on our cost model that can adaptively estimate the stopping level of the recursive division, minimizing the total computation cost, which have not been studied by previous works.

Set Cover Problem. The *set cover problem* (SCP) is a classical NP-hard problem, which targets at choosing a set of subsets to cover a universe set, such that the number of the selected subsets is minimized. SCP is actually a special case of our MS-SC problem, in which there exists only one spatial task. However, in most situations, we have more than one spatial task in the spatial crowdsourcing system. A variant of SCP is the *weighted set cover problem*, which associates each subset with a weight. The well-known greedy algorithm [22] can achieve an approximation ratio of $\ln(N)(\approx H(N)$ here $H(N) = \sum_{i=1}^{N} 1/i)$, where N is the size of the universe set. Sun and Li [21] studied *set cover games problem*, which covers multiple sets. However, they focused on designing a good mechanism to enable each single task to obtain a local optimal result. In contrast, our work aims to obtain a global optimal solution to maximize the score of assignment.

Different from SCP and its variants that cover only one universe set, our MS-SC problem is targeting to cover multiple sets, such that the assignment score is maximized. Furthermore, our MS-SC problem is also constrained by budget, time, and distance, which is much more challenging than SCP. To the

best of our knowledge, no prior works on SCP (and its variants) have studied the MS-SC problem, and existing techniques cannot be used directly to tackle the MS-SC problem.

9 Conclusion

In this paper, we propose the problem of the *multi-skill oriented spatial crowdsourcing* (MS-SC), which assigns the time-constrained and multi-skill-required spatial tasks with dynamically moving workers, such that the required skills of tasks can be covered by skills of workers and the assignment score is maximized. We prove that the processing of the MS-SC problem is NP-hard, and thus we propose three approximation approaches (i.e., greedy, g-D&C, and cost-model-based adaptive algorithms), which can efficiently retrieve MS-SC answers. Extensive experiments have shown the efficiency and effectiveness of our proposed MS-SC approaches on both real and synthetic data sets.

References

[1] Foursquare. https://foursquare.com.

[2] Google street view. https://www.google.com/maps/views/streetview.

[3] Taskrabbit. https://www.taskrabbit.com.

[4] Waze. https://www.waze.com.

[5] F. Alt, A. S. Shirazi, A. Schmidt, U. Kramer, and Z. Nawaz. Location-based crowdsourcing: extending crowdsourcing to the real world. In *NordiCHI 2010: Extending Boundaries*, 2010.

[6] A. Belussi and C. Faloutsos. Self-spacial join selectivity estimation using fractal concepts. *TOIS*, 16(2):161–201, 1998.

[7] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W.-C. Tan. Asking the right questions in crowd data sourcing. In *ICDE 2012*.

[8] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask?: jury selection for decision making tasks on micro-blog services. *VLDB 2012*, 5(11).

[9] Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, and Y. Tong. gmission: A general spatial crowdsourcing platform. *VLDB 2014*, 7(13).

[10] P. Cheng, X. Lian, Z. Chen, R. Fu, L. Chen, J. Han, and J. Zhao. Reliable diversity-based spatial crowdsourcing by moving workers. *VLDB 2015*, 8(10).

[11] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, and M. Shin. Anonysense: privacy-aware people-centric sensing. *MobiSys 2008*.

[12] D. Deng, C. Shahabi, and U. Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *SIGSPATIAL GIS 2013*.

[13] J. Fan, G. Li, B. C. Ooi, K.-l. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *ACM SIGMOD 2015*.

[14] Z. B. G. L. J. F. Huiqi Hu, Yudian Zheng and R. Cheng. Crowd-sourced poi labelling: Location-aware result inference and task assignment. *ICDE 2016*.

[15] S. S. Kanhere. Participatory sensing: Crowdsourcing data from mobile smartphones in urban spaces. In *MDM 2011*.

[16] R. M. Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[17] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *SIGSPATIAL GIS 2012*.

[18] S. H. Kim, Y. Lu, G. Constantinou, C. Shahabi, G. Wang, and R. Zimmermann. Mediaq: mobile multimedia management system. In *ACM MMSys 2014*.

[19] X. Liu, Q. He, Y. Tian, W.-C. Lee, J. McPherson, and J. Han. Event-based social networks: linking the online and offline social worlds. In *ACM SIGKDD 2012*.

[20] J. P. Rula, V. Navda, F. E. Bustamante, R. Bhagwan, and S. Guha. No one-size fits all: Towards a principled approach for incentives in mobile crowdsourcing. In *HotMobile 2014*. ACM.

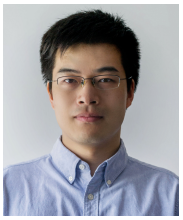
[21] Z. Sun, X.-Y. Li, W. Wang, and X. Chu. Mechanism design for set cover games when elements are agents. In *Algorithmic Applications in Management*, pages 360–369. Springer, 2005.

[22] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[23] D. Yang, G. Xue, X. Fang, and J. Tang. Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing. *MobiCom 2012*.

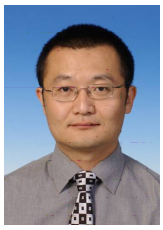
[24] Y. Zheng, R. Cheng, S. Maniu, and L. Mo. On optimality of jury selection in crowdsourcing. In *EDBT 2015*.

[25] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng. Qasca: A quality-aware task assignment system for crowdsourcing applications. In *ACM SIGMOD 2015*.



Peng Cheng received his BS degree and MA degree in Software Engineering in 2012 and 2014, from Xi'an Jiaotong University, China. He is now a PhD student in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. His research interests include crowdsourcing and spatial crowdsourcing.

Xiang Lian received the BS degree from the Department of Computer Science and Technology, Nanjing University, and the PhD degree in computer science from the Hong Kong University of Science and Technology. He is now an assistant professor in the Computer Science Department at the University of Texas Rio Grande Valley. His research interests include probabilistic/uncertain/inconsistent, uncertain/certain graph, time-series, and spatial databases.



Lei Chen received his BS degree in Computer Science and Engineering from Tianjin University, China, in 1994, the MA degree from Asian Institute of Technology, Thailand, in 1997, and the PhD degree in computer science from University of Waterloo, Canada, in 2005. He is now an associate professor in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. His research interests include crowdsourcing, uncertain and probabilistic databases, multimedia and time series databases, and privacy. He is a member of the IEEE.

Jinsong Han received his Ph.D. degree in computer science from Hong Kong University of Science and Technology in 2007. He is a member of CCF, ACM, and IEEE. His research interests focus on mobile computing, RFID, and wireless network.

Jizhong Zhao received his Ph.D. degree in computer science and technology from Xi'an Jiaotong University in 2001. He is a member of CCF, ACM, and IEEE. His research interests focus on computer software, pervasive computing, distributed systems, network security.