Destination-Aware Task Assignment in Spatial Crowdsourcing: A Worker Decomposition Approach

Yan Zhao, Kai Zheng, *Member, IEEE*, Yang Li, Han Su, Jiajun Liu, and Xiaofang Zhou, *Fellow, IEEE*

Abstract—With the proliferation of GPS-enabled smart devices and increased availability of wireless network, spatial crowdsourcing (SC) has been recently proposed as a framework to automatically request workers (i.e., smart device carriers) to perform location-sensitive tasks (e.g., taking scenic photos, reporting events). In this paper, we study a destination-aware task assignment problem that concerns the optimal strategy of assigning each task to proper worker such that the total number of completed tasks can be maximized whilst all workers can reach their destinations before deadlines after performing assigned tasks. Finding the global optimal assignment turns out to be an intractable problem since it does not imply optimal assignment for individual worker. Observing that the task assignment dependency only exists amongst subsets of workers, we utilize tree-decomposition technique to separate workers into independent clusters and develop an efficient depth-first search algorithm with progressive bounds to prune non-promising assignments. In order to make our proposed framework applicable to more scenarios, we further optimize the original framework by proposing strategies to reduce the overall travel cost and allow each task to be assigned to multiple workers. Extensive empirical studies verify that the proposed technique and optimization strategies perform effectively and settle the problem nicely.

Index Terms—Spatial crowdsourcing, spatial task assignment, algorithm

1 Introduction

The increased popularity of GPS-equipped smart devices and decreased cost of wireless mobile network (e.g., 4G network) have enabled people to move as sensors and participate some location-based tasks. Spatial crowdsourcing is a recently proposed concept and framework, which employs smart device carriers as workers to physically move to some specified locations and accomplish these tasks.

One of the main research problems in spatial crowdsourcing is how to assign tasks to workers strategically. Existing works focused on assigning tasks to workers to maximize the total number of completed tasks [13], the number of performed tasks for a worker with an optimal schedule [7], or the reliability and diversity score of assignments [5]. An implicit assumption shared by these work is that a worker can only or is willing to perform tasks that are close to her currently location (e.g., within a circle with given radius). While this is indeed realistic for many applications, we also observe some other scenarios where it is feasible for workers to perform tasks beyond her spatial vicinity. For instance, a worker who is driving on road towards a certain destination might not mind performing some tasks along the route as long as the extra travel cost (e.g., detour cost) does not affect her scheduled deadline at the destination. Note that, these tasks are not necessarily close to her original location so a specific valid range cannot be defined for each worker.

In this paper, we investigate the task assignment of spatial crowdsourcing under such a problem setting, namely Destination-aware Task Assignment (DATA). Specifically, given a user's current location, destination and deadline, before which she needs to arrive at the destination, it aims at finding an optimal assignment of tasks to workers such that the total number of task assignments is maximized. Note it actually consists of two sub-problems: 1) for each task, we need to assign it to the suitable workers; and 2) for each worker, we need to schedule a sequence by which a worker performs her assigned tasks. Compared to the previous work, the hardness of our problem lies in that, once the travel costs associated with moving to tasks' locations and the expiration time of tasks are taken into account, local optimal assignment does not

- *Y. Zhao is with the Institute of Artificial Intelligence, School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu 215168, China, and also with the Zhejiang Lab, Hangzhou, Zhejiang 310025, China. E-mail: zhaoyan@suda.edu.cn.*
- *K. Zheng and H. Su are with the University of Electronic Science and Technology of China, Chengdu, Sichuan 610054, China. E-mail: {zhengkai, hansu}@uestc.edu.cn.*
- *Y. Li is with the School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu 215168, China. E-mail: graberial@outlook.com.*
- *J. Liu is with the Renmin University of China, Beijing 100872, China. E-mail: jiajunliu@ruc.edu.cn.*
- *X. Zhou is with the University of Queensland, Brisbane, QLD 4072, Australia, and also with the Zhejiang Lab, Hangzhou, Zhejiang 310025, China. E-mail: zxf@itee.uq.edu.au.*

Manuscript received 8 Apr. 2018; revised 1 May 2019; accepted 6 June 2019. Date of publication 12 June 2019; date of current version 5 Nov. 2020. (Corresponding author: Kai Zheng.) Recommended for acceptance by H. Lee. Digital Object Identifier no. 10.1109/TKDE.2019.2922604

lead to global optimal result, that is assigning the most tasks to each worker does not necessarily imply the maximum number of accomplished tasks by all workers. The only existing work that considers task assignment and scheduling at the same time is [8], in which an approximate approach is developed that iteratively improves the assignment and scheduling to achieve more completed tasks. The second challenge is that the tasks reachable by each worker highly depend on the distance between origin and destination as well as the tightness of deadline. This makes pruning infeasible tasks more difficult than the conventional settings, which specify a valid range for each worker [8], [13].

We propose an exact solution that finds the optimal assignment result in terms of the total number of task assignments. The main idea of our approach is that, observing each worker only shares common tasks with a small portion of the entire worker set, we utilize a graph to represent the task dependency among workers (i.e., two workers sharing the same tasks have an edge in-between them) and apply a tree-decomposition procedure to divide all the workers into independent clusters. A top-down recursive search algorithm is then developed to traverse the tree in depth-first manner. In the meantime, dynamic upper and lower bounds are maintained during traversal in order to prune the tree nodes that cannot lead to optimal results. Compared to the iterative approach [8], our method finds the final and optimal assignment upon the completion of the search procedure, i.e., there is no re-matching and re-scheduling phase.

Although our previous work [27] has already achieved the optimization goal of maximizing the overall task assignments, it fails to consider the travel cost (i.e., in time or distance) of the workers during the assignment process, which is another critical factor since workers must physically go to the designated locations in order to perform the assigned tasks on spatial crowdsourcing platforms. However, the goals of maximizing the task assignment and minimizing the travel cost are often conflicting, which means optimizing both simultaneously could be difficult. To address this issue, in Section 4.1 of this extension, we incorporate a travel cost optimization strategy into the task assignment framework proposed in [27], which tries to minimize the overall travel cost of workers while keeping the number of task assignments unchanged by giving more priority to the performable task set with lower travel cost for each worker.

The second limitation of our previous study [27] is that it can only assign each task to a single worker. Nevertheless, some applications require each task to be assigned to multiple workers due to quality control purposes. Since allowing multiple workers to perform the same task (i.e., redundant task assignment mode) can affect the dependency relationship between workers during the independent worker partition phase, we carefully re-design our previous DATA solution in Section 4.2 to adapt it to the redundant task assignment mode by introducing new algorithms for both worker partition strategy and task assignment search strategy.

To summarize, our new technical contributions in this extension are five folds.

TABLE 1
Summary of Notations

Notation	Definition
s	Spatial task
l_s	Location of spatial task s
e_s	Expiration time of spatial task s
$maxW_s$	Maximum acceptable workers for task s
w	Worker
l_w	Current location of worker w
d_w	Destination of worker w
t_w	Deadline of worker w
$speed_w$	Movement speed of worker w
R	A task sequence
S_w	A task set for w
$VTS(w)$	A valid task set of w
$t(l)$	The arrival time of particular location l
$c(a, b)$	Travel distance from a to b
A	A spatial task assignment

1) We identify and study in depth two limitations in our previous DATA framework, which includes failing to consider travel cost factor and failing to support redundant task assignment.
2) We prove that the problems of Maximal Valid Task Set calculation and DATA are both NP-hard.
3) We incorporate a travel cost optimization strategy into the task assignment process, which tries to reassign workers the performable tasks with less travel cost whenever possible as long as the overall number of task assignment remains optimized.
4) We carefully re-design the worker partition algorithm and task assignment algorithm to make the DATA framework applicable to scenarios where each task should be assigned to multiple workers.
5) Extensive experiments are conducted to study the impact of the key parameters and effectiveness of our newly proposed algorithms. In particular, compared with the original exact task assignment approach, the travel cost optimization strategy can reduce the total travel cost by up to 24.35 percent, while the redundant task assignment strategy can improve the overall task assignments, which guarantees at least 41.3 percent tasks can be assigned to multiple workers in order to enhance the accuracy of task completion.

The remainder of this paper is organized as follows. Section 2 introduces the preliminary concepts and formulates the destination-aware task assignment problem. The proposed algorithms and related techniques are presented in Section 3, followed by the extension in Section 4. We report the results from empirical study in Section 5. Section 6 surveys the related work under different problem settings and Section 7 concludes this paper.

2 Problem Definition

In this section, we define a set of preliminaries in the context of self-incentivised single task assignment (i.e., a task can only be assigned to a worker) in spatial crowdsourcing with Server Assigned Tasks (SAT) mode [13]. Table 1 lists the major notations used throughout the paper.

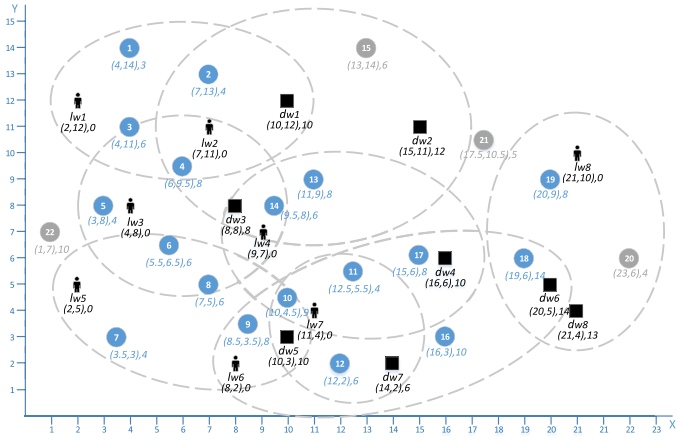


Fig. 1. Running example.

Definition 1 (Spatial Task). *A spatial task, denoted by* $s =< l_s, e_s >$*, is a task to be answered at location* l_s*, and will expire at* e_s*, where* $l_s : (x, y)$ *is a point in the 2D space.*

For simplicity and without loss of generality, we assume the processing time of each task is 0, which means that a worker will go to the next task upon finishing the current task.

Definition 2 (Worker). *A worker,* $w =< l_w, d_w, t_w, speed_w >$*, is a carrier of a mobile device who volunteers to perform spatial tasks. A worker can be in an either online or offline mode. A worker is offline when she is unable to perform tasks and is online when she is ready to accept tasks. An online worker is associated with her current location* l_w*, her destination location* d_w*, the deadline before when she must arrive at destination* t_w*, and her movement speed* $speed_w$*.*

Fig. 1 shows an example of several workers $W = \{w_1, w_2, \ldots, w_8\}$ and all tasks S (for simplicity, we just use the index to denote a specific task, i.e., $S = \{1, 2, \ldots, 22\}$) along and near the routes from l_{w_i} to $w_i.d$. Each worker with her current location, such as w_1 located at (2,12), starts from time zero; each task is associated with a location and deadline: Task 1 located at (4,14) will expire after 3 time units. For the sake of simplicity, we set the movement speed of each worker to 1 in this running example.

Definition 3 (Task Sequence). *Given an online worker* w *and a set of tasks assigned to her* S_w*, a task sequence on* S_w*, denoted as* $R(S_w)$*, represents the order by which* w *visits each task in* S_w*. The arrival time of* w *at task* $s_i \in S_w$ *(the time of completing task* s_i*) can be computed as follows:*

$$t_{w,R}(l_{s_i}) = \begin{cases} t_{w,R}(l_{s_{i-1}}) + c(l_{s_{i-1}}, l_{s_i})/speed_w & \text{if } i \neq 1 \\ c(l_w, l_{s_i})/speed_w & \text{if } i = 1, \end{cases} \quad (1)$$

where $c(a, b)$ *is the travel distance from location* a *to location* b*. The arrival time at destination after completing all tasks in* S_w *with the task sequence* R *is*

$$t_{w,R}(d_w) = t_{w,R}(l_{s_{|S_w|}}) + c(l_{s_{|S_w|}}, d_w)/speed_w. \quad (2)$$

When the context of w *and* R *is clear, we use* $t(l_{s_i})$ $(t(d_w))$ *to denote* $t_{w,R}(l_{s_i})$ $(t_{w,R}(d_w))$*.*

Definition 4 (Valid Task Set (VTS)). *A task set* S_w *is called a valid task set (VTS) for a worker* w*, if there exists a task sequence* $R(S_w)$*, such that,*

1) *all the tasks of* S_w *can be completed before their respective expiration time, i.e.,* $\forall s_i \in S_w, t(l_{s_i}) \leq e_{s_i}$*, and*
2) *the worker* w *can arrive at destination on time after completing all tasks in* S_w*, i.e.,* $t(d_w) \leq t_w$*.*

Definition 5 (Maximal Valid Task Set (MaxVTS)). *A Valid Task Set* S_w *is maximal if none of its super sets is still valid for a worker* w*.*

Note that there may exist more than one maximal VTS for a given worker w. In Fig. 1, $\{4, 13\}$, $\{14, 13\}$ and $\{4, 14, 13\}$ are valid task sets for worker w_2, but $\{2, 4\}$ is not a valid task set since w_2 cannot arrive at $w_2.d$ on time after finishing task 2 and 4. Note that neither $\{4, 13\}$ nor $\{14, 13\}$ is a maximal VTS since it is contained by $\{4, 14, 13\}$.

The MaxVTS calculation problem can be proved to be NP-hard by reduction from a Destination-aware Traveling Salesman Problem (DTSP). In the following, we give the definition of DTSP and prove it as NP-Complete.

Definition 6 (Destination-aware Traveling Salesman Problem (DTSP)). *Given a complete graph* $G(V, E)$ *with weight function* c*:* $V \times V \rightarrow Z$*, a source vertex* a*, a destination vertex* b *and cost* $k \in Z$*, where* $k \geq c(a, x) + c(x, b)$ *for any* $x \neq a$ *and* $x \neq b$*, the DTSP problem* $< G, c, a, b, k >$ *is to determine whether there exists a tour which visits each vertex exactly once, starting from the source vertex* a *and finishing at the destination vertex* b *with the cost of at most* k*.*

Lemma 1. *The DTSP problem is NP-Complete.*

Proof 1. The proof is shown in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/TKDE.2019.2922604. □

Lemma 2. *Given a worker* w *(with her current location* l_w*, destination location* d_w *and deadline* t_w*), s set of* n *tasks* S *and number* m*, deciding whether there exists a valid task sequence* R *(by which worker* w *has to start from* l_w *and end at* d_w *before* t_w*), st.* $|R| = m$*, is NP-Complete. That is, the decision problem of MaxVTS calculation is NP-Complete.*

Proof 2. The proof is shown in Appendix B, available in the online supplemental material. □

Since we have proved that the decision version of MaxVTS calculation problem is NP-Complete, we can conclude that the MaxVTS calculation problem is NP-hard.

Definition 7 (Spatial Task Assignment). *Given a set of workers* W *and a set of tasks* S*, a spatial task assignment, denoted by* A*, consists of a set of* $< worker, VTS >$ *pairs in the form of* $< w_1, VTS(w_1) >$*,* $< w_2, VTS(w_2) >$*,...,* $< w_{|W|}, VTS(w_{|W|}) >$*, where* $VTS(w_1) \cap VTS(w_2) ... \cap VTS(w_{|W|}) = \emptyset$*.*

Let $A.S$ denote the set of tasks that are assigned to all workers, i.e., $A.S = \cup_{w \in W} S_w$ and $\mathbb{A}$ denote all possible ways of assignments. The problem investigated in our paper can be formally stated as follows.

Problem Statement: Given a set of workers W and a set of tasks S, the Destination-aware Task Assignment ($DATA$)

problem aims to find the global optimal assignment A_{opti}, such that $\forall\, A_i \in \mathbb{A}$, $|\, A_i.S\,| \leq |\, A_{opti}.\,|$.

Lemma 3. *The DATA problem is NP-hard.*

Proof 3. The proof is shown in Appendix C, available in the online supplemental material. □

3 Algorithm

Since the DATA problem is NP-hard, a simple greedy algorithm is to use the maximum valid task set for each worker as the assignment result. This can hardly be a satisfying result since multiple workers may be assigned the same set of tasks that may leave more tasks unassigned. In this paper, we develop an exact solution with three steps. First, we devise a dynamic programming algorithm to find the set of maximal valid task sequences for each worker. It can be shown that the global optimal result is the union of one possible valid task sequence of all workers. Second, to avoid exhaustive search through all the possible combination of valid task sequences, we utilize a tree-decomposition technique to separate all workers into independent clusters and organize them into a tree structure, such that the workers in sibling nodes of the tree do not share the same valid tasks. In the final step, the tree is traversed in depth-first manner to find the optimal assignment. During the traversal, a lower bound that indicates the minimal number of required tasks for each sub-tree is dynamically maintained and compared against its upper bound (i.e., the maximum number of tasks that can be assigned to the sub-tree). If the lower bound is greater than the upper bound, the sub-tree can be eliminated without further exploration.

3.1 Valid Task Set Generation

3.1.1 Finding Reachable Tasks

Due to the constraint of workers' deadlines and tasks' expiration time, each worker can only complete a small subset of tasks. Therefore, we first find the set of tasks that can be reached by each worker without violating the constraints. The reachable task subset for a worker w, denoted as RS_w, should satisfy the following two conditions: $\forall s \in RS_w$,

(1) $c(l_w, l_s) \leq e_s$ and
(2) $c(l_w, l_s) + c(l_s, d_w) \leq t_w$.

The above two conditions guarantee that a worker can travel from her origin to the location of task s directly before it expires and still have sufficient time to arrive her destination before deadline. From computational perspective, the reachable tasks (satisfying the condition 1) fall inside an ellipse with the worker's origin and destination as focus and the maximum travel distance (i.e., $t_w \times speed_w$) as the length of major axis. It is easy to see the time complexity is $O(|W| \cdot |S|)$, where $|W|$ and $|S|$ are the numbers of workers and tasks respectively. In Fig. 1, the blue numbered circles denote all the reachable tasks and the grey ones represent the unreachable tasks.

3.1.2 Finding Maximal Valid Task Set

Given the reachable task set for each worker, we next find the set of MaxVTS, which is shown to be an NP-hard problem in Lemma 2. However, the reachable task set for each worker is usually not large, which means this problem can still be solved by an efficient algorithm in practice. Moreover, finding the MaxVTSs for each worker is completely independent and can be easily parallelized.

In the sequel, we present a dynamic programming algorithm that iteratively expands the sets of tasks in the ascending order of set size and find all MaxVTSs in each iteration. For each task in one set, we consider the scenario that it is finished in the end, and find all completed task sequences. Specifically, given a worker w, and a set of tasks $Q \subseteq RS_w$. We define $\text{opt}(Q, s_j)$ as the maximum number of tasks completed by scheduling all the tasks in Q with constraints starting from l_w and ending at l_{s_j}, and R as the corresponding task sequence on Q to achieve this optimum value. We also use s_i to denote the second-to-last task before arriving at s_j in R, and R' to denote the corresponding task sequence for $\text{opt}(Q - \{s_j\}, s_i)$. Then $\text{opt}(Q, s_j)$ can be calculated by

$$opt(Q, s_j) = \begin{cases} 1 & \text{if } |Q| = 1 \\ \max_{s_i \in Q, s_i \neq s_j} opt(Q - \{s_j\}, s_i) + \delta_{ij} & \text{otherwise,} \end{cases} \tag{3}$$

$$\delta_{ij} = \begin{cases} 1 & \text{if } t(l_{s_j}) \leq e_{s_j}, and\ t(l_{s_j}) + c(l_{s_j}, d_w) \leq t_w \\ 0 & \text{otherwise,} \end{cases},$$

where δ_{ij} is an indication function, in which $\delta_{ij} = 1$ means s_j can be finished after appending s_j to R' and the worker can arrive the destination before her deadline.

Algorithm 1. MaxVTS

```
1   Input: w, RS_w
    Output: Q_w
2   Q_w ← null;
3   for each task s_i in RS_w = {s_1, s_2, ..., s_n} do
4     opt({s_i}, s_i) ← 1;
5     Q_w ← Q_w ∪ {{s_i}};
6     pre({s_i}, s_i) ← null;
7   for len ← 2 to n do
8     for each subset Q ⊆ RS_w of size len do
9       for each s_j ∈ Q do
10        opt(Q, s_j) ← max_{s_i∈Q, s_i≠s_j} opt(Q − {s_j}, s_i) + δ_ij;
11        pre(Q, s_j) ← arg max_{s_i∈Q, s_i≠s_j} opt(Q − {s_j}, s_i) + δ_ij;
12        if δ_ij = 1 then
13          Q_w ← Q_w ∪ {Q};
14          for each Q' ∈ Q_w do
15            if Q' ⊂ Q then
16              Remove Q';
17  compute R* based on opt and pre;
18  return Q_w
```

When Q contains only one task s_i, the problem is trivial and $\text{opt}(\{s_i\}, s_i)$ is set to 1. When $|Q| > 1$, we need to search through Q to examine all possibilities of valid task sets and find the particular s_i that achieves the optimum value of $opt(Q, s_j)$. Algorithm 1 outlines the structure of this procedure. Note that we use $pre(Q, s_j)$ to record the last-to-second task s_i before achieving $opt(Q, s_j)$ to facilitate the reconstruction of the optimal valid task sequence R. After initialization, the algorithm generates and processes sets in the increasing order of their size from 2 to n (lines 7-8). For

TABLE 2
Maximal Valid Task Sets

W	Maximal Valid Task Sets	$\|maxS\|$
w_1	$\{s_1, s_2\}, \{s_2, s_3\}, \{s_3, s_4\}$	2
w_2	$\{s_2\}, \{s_4, s_{13}, s_{14}\}$	3
w_3	$\{s_{14}\}, \{s_6, s_8\}, \{s_5, s_6\}, \{s_4, s_6\}, \{s_4, s_5\}, \{s_3, s_4\}$	2
w_4	$\{s_{10}, s_{17}\}, \{s_{11}, s_{17}\}, \{s_{13}, s_{14}, s_{17}\}$	3
w_5	$\{s_6, s_8\}, \{s_7, s_9\}, \{s_8, s_9\}, \{s_8, s_{10}\}, \{s_9, s_{10}\}$	2
w_6	$\{s_{10}, s_{16}\}, \{s_9, s_{10}, s_{18}\}, \{s_9, s_{12}, s_{16}\}, \{s_{12}, s_{16}, s_{18}\}$	3
w_7	$\{s_{10}\}, \{s_{11}\}, \{s_{12}\}$	1
w_8	$\{s_{18}, s_{19}\}$	2

each task $s_j \in Q$, it computes $opt(Q, s_j)$ and $pre(Q, s_j)$ according to Equation (3) (lines 10-11). Finally, whenever Q can be added to Q_w, we remove its proper subsets that already exist in Q_w (lines 14-16). To save space, the procedure of constructing R^* from tables opt and pre is omitted here.

Algorithm 1 correctly computes $MaxVTS$ set with $O(2^{|RS_w|} \cdot |RS_w|^3)$ time complexity, where $|RS_w|$ is the number of reachable tasks for worker w. Table 2 shows the MaxVTSs of workers based on Equation (3) and their maximum number of completable tasks $|maxS|$.

3.2 Worker Partition

The main computational challenge lies in huge search space when enumerating all possible combinations of the valid task sets of each worker, which increases exponentially with respect to the number of workers. However, in practice a worker shares the same tasks with only a few other workers who have similar or intersected travel routes.

Definition 8 (Worker Dependency). *Given two workers w_i, w_j, and their respective reachable task sets, RS_{w_i}, RS_{w_j}, they are independent with each other if $RS_{w_i} \cap RS_{w_j} = \emptyset$. Otherwise, they are dependent with each other.*

For instance, in Fig. 1, w_1 has dependency with w_2 and w_3, but is independent with the rest of workers. In our work, we aim to leverage the independency amongst workers and partition the worker set into independent groups, such that the optimal assignment can be found more efficiently in each groups.

3.2.1 Worker Dependency Graph

Given a worker set W and task set S, we can construct a Worker Dependency Graph (WDG) $G(V, E)$, where each node $v \in V$ represents a worker $w_v \in W$. An edge $e(u, v) \in E$ exists between u and v if the two workers w_u and w_v are dependent with each other. The time complexity of WDG construction is $O(|W|^2 \cdot |RS|)$, where $|RS|$ is the average number of reachable takes for each worker. Fig. 2a illustrates the WDG for worker set shown in Fig. 1.

3.2.2 Graph Partition

In this part, we need to decompose the dependency relationship by partitioning WDG. To this end, we utilize the notion of tree-decomposition [16], which transforms a graph into a tree structure.

Definition 9 (Tree Decomposition). *Given an undirected graph $G = (V, E)$ composed of a set V of vertices and a set E of edges. A tree-decomposition of G is a pair $(\mathbb{X}, T)$, where $\mathbb{X} = \{X_1, \ldots, X_n\}$ is a family of subsets of V, and T is a tree whose nodes are the subsets X_i, satisfying the following properties [16]:*

1) $\cup_{i \in n} X_i = V$, *and*
2) $\forall (v, w) \in E, \exists X_i \in \mathbb{X}$ *containing both v and w, and*
3) *if X_i, X_j and X_k are nodes, and X_k is on the path from X_i to X_j, then $X_i \cap X_j \subseteq X_k$.*

The tree decomposition of a graph is far from unique. Next, we briefly introduce some related concepts and then describe the maximum cardinality search (MCS) algorithm [20] to find the tree-decomposition.

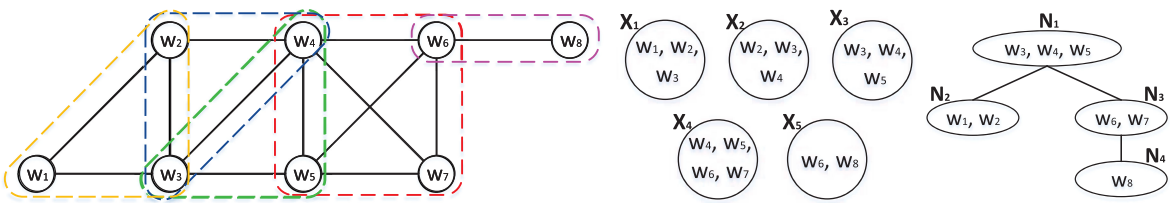
Definition 10 (Chordal Graph). *A graph is a chordal graph if every cycle of length > 3 has a chord, i.e., edge joining two non-consecutive vertices of a cycle [1].*

Definition 11 (Maximal Clique). *Every maximal clique of a chordal graph $G = (V, E)$ is of the form $\{v\} \cup C(v)$, for some vertex $v \in V$ where $C(v) = \{w | (v, w) \in E, \delta(v) < \delta(w)\}$. δ is a perfect elimination ordering of vertices in a graph such that, for each vertex v, v and the neighbors of v that occur later than v in the order form a clique [17].*

Property 1. *A tree-decomposition of a chordal graph consists of the set of its maximal cliques [20].*

The MCS algorithm consists of following steps.

1) Given a WDG, construct the corresponding chordal graph by adding suitable new edges.
2) Find δ on the derived chordal graph.
3) Identify the maximal cliques in the chordal graph. For each vertex v of δ, the maximal clique containing v is a graph with the nodes $\{v\} \cup C(v)$.
4) The maximal cliques will be the nodes (i.e., $\mathbb{X}$) of the tree-decomposition result.



(a) Worker Dependency Graph (b) Graph Partition (c) Tree Construction

Fig. 2. Worker partition.

The time complexity for the above algorithm is $O(|V| + |E'|)$, where E' is the number of edges in the chordal graph obtained. Take the WDG shown in Fig. 2a as an example. Since it is already a chordal graph according to its definition, we do not need to take any actions in the first step. In the second step, we find the perfect elimination ordering of graph G and sort all the nodes by δ in ascending order: $\{w_8, w_1, w_2, w_7, w_3, w_5, w_6, w_4\}$. Lastly, the maximal cliques of the graph can be found and output as the nodes of tree-decomposition: $\mathbb{X} = \{\{w_1, w_2, w_3\}, \{w_2, w_3, w_4\}, \{w_3, w_4, w_5\}, \{w_4, w_5, w_6, w_7\}, \{w_6, w_8\}\}$, as shown in Fig. 2b.

3.2.3 Tree Construction

According to the definition of tree-decomposition, if two nodes do not share the same vertexes, the workers belonging to the two nodes are independent with each other. In this step, our goal is to organize the subsets of workers in a tree structure such that the sibling nodes are independent with each other. Facilitated by such a tree structure, we can solve the optimal assignment sub-problem on each sibling node independently. Since the search cost is largely affected by the number of workers, we would like to make the tree as balanced as possible, i.e., to avoid any node with significantly more workers than the others. To this end, we devise the following Recursive Tree Construction (RTC) algorithm:

1) Try to remove the vertices in each node $X_i \in \mathbb{X}$ (output in the graph partition step) from the WDG G. G will be separated into a few components, of which the largest one is recorded as G_{max}. $|G_{max}|$ is the number of vertices in G_{max}.
2) Pick the node X_{min} that leads to the least $|G_{max}|$ upon the completion of the previous loop (pick the smallest X_i as X_{min} when there is a tie on $|G_{max}|$ and randomly pick a X_i as X_{min} when their cardinalities are same). Set X_{min} as the parent node for each output of the recursive procedure in step 3.
3) Apply the MCS algorithm on each sub graph by removing workers of X_{min} and recursively perform this algorithm on the output of MCS algorithm.
4) Return $N = X_{min}$ as the root node of this sub-tree.

We can derive from the RTC algorithm that the balanced tree (constructed from $G(V, E)$), denoted by $\mathcal{T}$ (with a set of nodes $N_{\mathcal{T}} = \{n_1, n_2, \ldots, n_{|N_{\mathcal{T}}|}\}$), satisfies the following properties:

1) $\cup_{i \in |N_{\mathcal{T}}|} n_i = V$, and
2) for each node $n_i \in N_{\mathcal{T}}$, removing n_i from $G_{\mathcal{T}_i}$ leads to the least $|G^{max}_{\mathcal{T}_i}|$, where $\mathcal{T}_i$ is the subtree rooted with node n_i, $G_{\mathcal{T}_i}$ is the WDG for workers in the subtree $\mathcal{T}_i$ and $G^{max}_{\mathcal{T}_i}$ is the largest subgraph by removing n_i from $G_{\mathcal{T}_i}$, and
3) workers in the subtrees rooted with sibling nodes are independent with each other.

The time complexity of RTC in the ith recursion is $O(|\mathbb{X}^i| + |G^i_{sub}| \cdot (|V^i| + |E^i|))$ (including finding node X^i_{min} from $\mathbb{X}^i$ and applying MCS algorithm in each subgraph), where G^i_{sub} is the subgraph set by performing step 1 in the ith recursion. Thus the total time complexity of RTC is $O(\sum_{i=1}^{m}(|\mathbb{X}^i| + |G^i_{sub}| \cdot (|V^i| + |E^i|)))$, where m is the number of recursions. Constructing the tree with nodes in Fig. 2b by RTC algorithm, we get the final tree structure, which is illustrated in Fig. 2c.

3.3 Search

In this section, we present our search algorithm framework for using a tree-decompositon to solve the $DATA$ problem.

Once the worker dependency graph has been transformed to a tree structure, the optimal assignment can be found by a depth-first search through the tree. First of all, we give an overview of whole process including the previous steps in Algorithm 2.

Given the worker set W and task set S, the reachable task set RS_w and maximal valid task sets Q_w are computed for each worker w (line 2-4), and the corresponding worker dependency graph G is constructed (line 5). Then for each connected component $g \in G$, we decompose g into a set of vertex clusters with MCS algorithm (line 7) and organize them into a tree with RTC algorithm (line 8). Lastly the depth-first search algorithm ($DFSearch$) is invoked on each tree to find the optimal assignment in each sub problem. Since each component of G is independent with each other, the final result is to simply sum up the optimal assignment of each component g (line 9).

Algorithm 2. Solution Overview

```
    Input: W, S
    Output: Opt
1   Opt ← 0; Q' ← ∅; S' ← S;
2   for each worker w ∈ W do
3     RS_w ← compute the reachable tasks for w;
4     Q_w ← MaxVTS(w, RS_w);
5   G ← construct worker dependency graph;
6   for each connected component g ∈ G do
7     X_g ← decompose g into vertex clusters;
8     N_g ← organize X_g into a tree;
9     Opt ← Opt + DFSearch(N_g, S, W_{N_g}, LB(N_g));
10  return Opt;
```

Next we elaborate the details of $DFSearch$ procedure in Algorithm 3. The procedure takes four parameters: the root node N of the sub-tree to be traversed, the remaining unassigned task set S, the remaining available workers W_N in node N and a heuristic h indicating the minimum required number of tasks yet to be assigned in order to beat the current optimal assignment.

The algorithm starts with computing an upper bound $UB(N)$ of the number of tasks that can be assigned to the workers contained in the sub-tree rooted with N (line 2), and compares it against with the heuristic h that represents a lower bound $LB(N)$ of the number of tasks that need to be assigned to this sub-tree in order to beat the optimal assignment Opt found so far (line 3). Obviously, if $UB < LB$, this sub-tree can be safely pruned since it cannot lead to a better assignment. The ways to derive $UB(N)$ and $LB(N)$ will be discussed in Sections 3.3.1 and 3.3.2.

Then the algorithm branches depending on W_N, the worker set contained by current node N. If there are still workers to be probed (line 5), we will sequentially examine each available worker in W_N (line 6), get a new maximal VTS (Q_w) by eliminating the assigned tasks (Q') from the the existing maximal VTS (line 7), and then recursively call the

$DFSearch$ procedure by passing in the updated remaining task set $(S - Q)$, updated worker set $(W_N - w)$ and the updated heuristic $(h - |Q|)$. The optimal assignment is updated if the returned assignment plus $|Q|$ (i.e., the number of tasks assigned to current examined worker) is greater (line 10). The heuristic value h is updated accordingly since a better assignment is just found (line 11). On the other hand, if all the workers have been enumerated (line 13), the algorithm will invoke $DFSearch$ procedure on each child node of N. Since each child node (and their sub-tree) is independent with each other as guaranteed by our tree-decomposition algorithm in previous phase, the problem of finding optimal assignment for each sub-tree can be solved independently, and then summed up to obtain the global optimal assignment (line 14-15). The time complexity of Algorithm 3 is $O(\sum_{i=1}^{r}(|W_N^i| \cdot |Q_w^i| + |N_{child}^i|))$, where r is the number of recursions, $|W_N^i|$ is the number of workers in the node W_N^i in the ith recursion, $|Q_w^i|$ is the number of MaxVTSs of worker w in the ith recursion, and $|N_{child}^i|$ is the number of child nodes of N in the ith recursion.

Accuracy. In the worker partition phase, we separate all workers into independent clusters and organize them into a tree structure, where workers in sibling nodes of the tree do not share the same valid tasks. Algorithm 3 illustrates that, given any non-empty node N of the tree, we check all the task assignments for all the workers (contained in the sub-tree whose root is N) and their valid task sets, thus the optimal task assignment with maximal number of assigned tasks can be found in this subtree rooted with N (line 5-12). Since workers in sibling nodes do not share the same valid tasks, we can simply sum up the task assignment results in the subtrees rooted with sibling nodes in order to get the global optimal task assignment (line 14-15).

Algorithm 3. DFSearch

```
    Input: N, S, W_N, h
    Output: Opt
1   Opt ← 0;
2   UB(N) ← compute the upper bound of assigned tasks for
    the sub-tree rooted with N;
3   if UB(N) < h then
4     return0;
5   if W_N ≠ ∅ then
6     for each worker w ∈ W_N do
7       Q_w ← Q_w − Q';
8       for each maximal valid task set Q ∈ Q_w do
9         Q' = Q' ∪ Q;
10              Opt ← max{DFSearch(N, S − Q, W_N − w, h−
                |Q|) + |Q|, Opt};
11        h ← Opt;
12        Q' ← ∅;
13  else
14    for each child node N_i of N do
15      Opt+ ← DFSearch(N_i, S, W_{N_i}, LB(N_i));
16  return Opt;
```

3.3.1 Upper Bound Estimation

The upper bound of a node N, denoted as $UB(N)$, represents the maximum number of tasks that can be finished by the sub-tree rooted at N. A simple estimation of $UB(N)$ is to sum up the cardinality of the maximum valid task set of each worker in this sub-tree, i.e.,

$$UB(N) = \sum_{i=1}^{|W|}(|maxS_{w_i}|), \tag{4}$$

where W denotes all the workers in the current sub-tree, and $maxS_{w_i}$ denotes the maximum valid task set that can be finished by w_i. $maxS_{w_i}$ can be obtained by choosing the maximal valid task set of w_i with the greatest cardinality, i.e., $maxS_{w_i} = \max\{Q|Q \in MaxVTS(w_i, S)\}$.

For example, when the search algorithm reaches N_3 in Fig. 2c, $UB(N_3)$ can be estimated as follows:

$$UB(N_3) = |maxS_{w_6}| + |maxS_{w_7}| + |maxS_{w_8}| = 3 + 1 + 2 = 6,$$

where the $|maxS|$ can be looked up from Table 2.

Lemma 4. *$UB(N)$ upper bounds the optimal assignment of the sub-tree rooted at N, and the bound is tight.*

Proof 4. Since in any task assignment A, including the optimal one, the number of tasks completed by each worker w cannot exceed $|maxS_w|$, the following inequality always holds:

$$|A.S| = |\cup_{w \in W} S_w| \leq \sum_{w \in W} |S_w| \leq \sum_{w \in W} |maxS_w| = UB(N).$$

When all the workers in N are independent with each other, i.e., they do not share any task, the optimal assignment will be equal to $UB(N)$ since the above inequality becomes

$$|\cup_{w \in W} S_w| = \sum_{w \in W} |S_w| = \sum_{w \in W} |maxS_w|.$$

Therefore the upper bound is tight. □

3.3.2 Lower Bound Estimation

In order to prune the unpromising branch as early as possible, we also calculate an lower bound heuristic $LB(N)$ and pass it as a parameter h to the recursive procedure on the child node. $LB(N)$ implies the minimal number of tasks that must be completed by the workers in the subtree rooted at N in order to find a better assignment than the current best one. Obviously, whenever the upper bound of one node is less than the lower bound, its sub-tree can be discarded safely since none of the possible assignment on the workers in this sub-tree can lead to better global assignment.

Now let us describe how to estimate the lower bound of N_i as the child node of N. Suppose N has m child nodes, i.e., $N_1, N_2, \ldots, N_m$. The $DFSearch$ algorithm will invoke the procedure on each child node and try to get a better assignment Opt for each node before returning to its parent node N (line 14). We can estimate the lower bound of N_i by the following formula,

$$LB(N_i) = h - \sum_{j=1}^{i-1} Opt(N_j) - \sum_{j=i+1}^{m} UB(N_j), \tag{5}$$

where (1) h is the minimal number of tasks required for the workers in all child nodes of N (i.e., $N_1, N_2, \ldots, N_m$); (2) $\sum_{j=1}^{i-1} Opt(N_j)$ represents the optimal assignment of the sub-

trees that have already been traversed by the procedure; (3) $\sum_{j=i+1}^{m} UB(N_j)$ represents the upper bound of the optimal assignment of the sub-trees that are yet to be probed.

Lemma 5. *$LB(N_i)$ is the minimal number of tasks to be completed by workers in the sub-tree rooted at N_i.*

Proof 5. If the optimal number of tasks assigned to N_i is less than $LB(N_i)$, then the maximum number of tasks that can be completed by workers in all child nodes of N will be less than h:

$$\begin{aligned}\sum_{j=1}^{m} Opt(N_j) &= Opt(N_i) + \sum_{j=1}^{i-1} Opt(N_j) + \sum_{j=i+1}^{m} Opt(N_j) \\ &< LB(N_i) + \sum_{j=1}^{i-1} Opt(N_j) + \sum_{j=i+1}^{m} UB(N_j) \\ &= h.\end{aligned}$$

Therefore in order to complete at least h tasks, the optimal assignment of N_i must satisfy $Opt(N_i) \geq LB(N_i)$. □

When invoking the $DFSearch$ procedure for the first time (line 10 of Algorithm 2), the lower bound of the entire tree can be estimated by the assignment of a greedy algorithm, i.e., the union of maximal valid task sets of all workers.

3.3.3 Optimization

In this part we briefly discuss some optimization schemes to further reduce the search cost.

1) Re-ordering sub-tree traversal: before invoking the $DFSearch$ procedure on the sub-trees of a node, we first sort all the sub-trees in the ascending order of the number of their associated workers. The rationality behind this optimization is that, the impact of loose pruning bounds in the very beginning to search performance can be reduced when applying on a sub-tree with less workers, while the tighter bounds can make the search on larger sub-trees more efficient. We update $DFSearch$ algorithm by sorting all the sub-trees in the ascending order of the number of their associated workers before line 14 of Algorithm 3.
2) Optimization of UB: we can further improve the upper bound of each sub-tree by applying a modified version of $DFSearch$ on the whole tree in bottom-up manner. The only difference lies in that, when searching the sub-trees, the input task set S is always the entire task set. Essentially this process finds the optimal assignment for workers in each sub-tree by assuming they can access all tasks (while they cannot in fact due to existence of other dependent workers). It is worth noting that the extra overhead incurred by this optimization is minimal, since the "optimal result" of a sub-tree is now independent with its parent node. Therefore when invoking $DFSearch$ from leaf nodes all the way up to the root, we can record the return value as UB on each node, and its parent node can simply use this record to derive its upper bound without recursively applying this procedure again. We can update $DFSearch$ algorithm by using "$Opt+ \leftarrow DFSearch(N_i, S', W_{N_i}, LB(N_i))$" to replace line 15 of Algorithm 3, where S' is the entire task set initialized in line 1 of Algorithm 2. The return value is the upper bound of assigned tasks for the sub-tree rooted with N. We compute the upper bound for each node N from bottom up.

TABLE 3
Time Complexity of the DATA Algorithm

Operation	Complexity
Valid Task Set Generation	$O(\|W\| \cdot \|S\| + \|W\| \cdot 2^{\|RS\|} \cdot \|RS\|^3))$
Worker Partition	$O(\|W\|^2 \cdot \|RS\| + \sum_i^m (\|\mathbb{X}^i\| + \|G^i_{sub}\| \cdot (\|V^i\| + \|E'^i\|)))$
Search	$O(\sum_i^r (\|W^i_N\| \cdot \|Q^i_w\| + \|N^i_{child}\|))$

3.4 Limitation of DATA

Our DATA problem requires each worker to specify her destination and deadline when she is ready to perform tasks. As its name (i.e., Destination-aware Task Assignment) suggests, this problem can only be applied in the destination-aware scenarios. In this work, we assume the processing time of each task is 0, which is a common assumption in spatial crowdsourcing studies [7], [8] due to the fact that most existing spatial tasks (e.g., taking photos/videos) are simple enough to be completed instantaneously. However, our proposed algorithms can be extended to handle more complex spatial tasks. To this end, we can modify the arrival time in Equations (1) and 2 by adding the processing time of each task in the corresponding task sequence.

Though our algorithm can provide an exact solution, the calculation is relatively inefficient. Table 3 summarizes the cost of each operation of our algorithms. Clearly the cost is dominated by the MaxVTS generation phase with an exponential time complexity, which is computationally expensive when $|RS|$ is large. Therefore, our algorithm is not suitable for a task-dense area, i.e., each worker has a large number of reachable tasks. However, in practice, the algorithm is still efficient because the number (i.e., $|RS|$) of reachable tasks for each worker is a relatively small value. Moreover, since the MaxVTS generation of each worker is independent with each other, we can calculate the MaxVTSs for each worker in parallel to improve efficiency.

4 Extension

As the extension of our previous work [27], we will present in this section two optimizations that will reduce the overall travel cost and support redundant task assignment respectively.

4.1 Travel Cost Optimization Strategy

The problem with the original framework is that it only maximizes the number of task assignments, without considering the travel cost (e.g., in time or distance) of the workers during the assignment process. In the problem settings of spatial crowdsourcing, travel cost is also critical issue since workers must physically go to the location of the spatial task in order to perform it. To address this issue, we propose a strategy, referred to as Travel Cost Optimization

Strategy, to improve the overall task assignment by giving higher priority to the valid task sets with lower travel cost.

We first introduce the notion of valid route, by which a worker can travel from her original location to her destination passing a set of valid tasks. More formally,

Definition 12 (Valid Route). *Given a worker w and a valid task set assigned to her, VTS, there may exist more than one task sequences in VTS for w. Let $R_i(VTS)$ denote a task sequence on VTS, representing the order by which w visits each task in VTS, and $\mathbb{R}(VTS) = \{R_1(VTS), R_2(VTS), ...\}$ denote all possible task sequences on VTS. A route from the worker's original location to her destination passing all the tasks of VTS, $l_w \rightarrow VTS \rightarrow d_w$, is called a valid route for w, if there exists a task sequence $R_i(VTS)$, such that,*

1) *all the tasks of VTS can be completed before their respective expiration time, e.g., $\forall s_i \in VTS$, $t_{w,R_i(VTS)}(l_{s_i}) \leq e_{s_i}$, and*
2) *the worker w can arrive destination on time after completing all tasks in VTS, e.g., $t_{w,R_i(VTS)}(d_w) \leq t_w$, and*
3) *the arrival time at destination by following the task sequence $R_i(VTS)$ is minimal, e.g., $\forall R_j(VTS) \in \mathbb{R}(VTS)$, $t_{w,R_i(VTS)}(d_w) \leq t_{w,R_j(VTS)}(d_w)$.*

Intuitively, tasks which are closer to a worker have smaller travel costs. Therefore, we define the travel cost from a worker's location l_w to her destination d_w passing a valid task set VTS, in terms of the Euclidean distance of the corresponding valid route, denoted by $c(l_w, d_w, VTS)$. Consequently, by computing the distance among every worker, her performable spatial tasks (i.e., those in a valid task set), and her destination, we can associate higher priorities to the closer valid task sets. Moreover, given a set of workers W and a set of tasks S, we define the aggregate travel cost, denoted by $ac(W, S)$, as the sum of the Euclidean distances of the valid routes for all workers in W while satisfying the spatio-temporal constraints of workers and tasks.

Taking the worker w_8 and her valid task set $\{s_{18}, s_{19}\}$ in Fig. 1 as a case, worker w_8 can perform task s_{18} and s_{19} in turn, or she can successively perform task s_{19} and s_{18} before arriving at her destination. However, the valid route for w_8 based on the valid task set $\{s_{18}, s_{19}\}$ is the route from the location of w_8 to her destination passing task s_{19} and s_{18} in turn, since $t_{w_8,(s_{19},s_{18})}(dw_8)$ is less than $t_{w_8,(s_{18},s_{19})}(d_{w_8})$. Correspondingly, the travel cost for $\{s_{18}, s_{19}\}$ is the arrival time at the destination of w_8 by following the task sequence (s_{19}, s_{18}), i.e., $c(l_{w_8}, d_{w_8}, \{s_{18}, s_{19}\}) = t_{w_8,(s_{19},s_{18})}(d_{w_8})$.

With the knowledge of the travel cost of valid routes for all the workers, we incorporate the travel cost in the search process to maximize the task assignments while minimizing the travel cost of the workers whenever possible (line 13, 19 and 22 of Algorithm 4).

4.2 Redundant Task Assignment Strategy

Another problem with the proposed DATA solution is that it can just be applied for the single task assignment in spatial crowdsourcing, in which each spatial task is only assigned to one worker. The assumption here is that all the workers are trusted, and thus they complete the spatial tasks correctly without any malicious intentions [6], [12]. In practice, however, there inevitably exist some workers who either intentionally (e.g., malicious workers) or unintentionally (i.e., making mistakes) perform the tasks incorrectly (i.e., being dishonest about physically going to the locations of the spatial tasks), which cannot guarantee the quality of task completion. To tackle this problem, we improve the DATA solution by changing the single task assignment to the redundant task assignment [13], in which each spatial task can be completed by a few available workers in proximity of the task, such that majority voting can be applied to improve the quality of task completion. The intuitive assumption shared by the redundant task assignment is based on the idea of the wisdom of crowds [19] that the majority of the workers can be trusted and thus the validity of task results provided by a group of workers can be verified by majority voting.

TABLE 4
Available Worker Set

S	Available Worker Set	S	Available Worker Set
s_1	w_1	s_{10}	w_4, w_5, w_6, w_7
s_2	w_1, w_2	s_{11}	w_4, w_7
s_3	w_1, w_3	s_{12}	w_6, w_7
s_4	w_1, w_2, w_3	s_{13}	w_2, w_4
s_5	w_3	s_{14}	w_2, w_3, w_4
s_6	w_3, w_6	s_{16}	w_5
s_7	w_5	s_{17}	w_6
s_8	w_3, w_5	s_{18}	w_6, w_8
s_9	w_5, w_6	s_{19}	w_8

In detail, a task s, associated with its maximum capacity (e.g., maximum acceptance workers, $maxW_s$), can be performed by at most $maxW_s$ workers instead of being completed by a particular worker, where $maxW_s$ is specified by the requester who issues the task s. Clearly, the higher the $maxW_s$ value is, the more chance that the task is completed correctly. To apply the DATA solution to the redundant task assignment problem, for each task s, we first calculate the workers who are allowed to perform it, namely Available Worker Set (AWS), and correspondingly $|AWS|$ denotes the number of available workers for s. Then we re-define the worker dependence based on AWS.

To obtain the available workers for each task, we employ the inverted file to improve the retrieval speed. The Available Worker Set of each task in our running example is illustrated in Table 2.

Based on the AWS for each task, we now re-define the notion of worker dependency.

Definition 13 (Worker Dependency). *Given two workers w_i, w_j, and their respective reachable task sets, RS_{w_i}, RS_{w_j}, they are independent with each other if either of the following conditions is satisfied:*

1) $RS_{w_i} \cap RS_{w_j} = \emptyset$, *or*
2) $\forall s \in \{RS_{w_i} \cap RS_{w_j}\}, |AWS(s)| \leq maxW_s$.

Otherwise, they are dependent with each other.

Consider the running example in Fig. 1 and set the maximum number of acceptance workers of s_{18} to 3, i.e., $maxW_{s_{18}} = 3$. w_8 is independent with w_6 since the available workers' number of their shared task (i.e., s_{18}) is 2, which is less than $maxW_{s_{18}}$. w_8 also has independency with the rest of workers because w_8 shares no reachable tasks with them.

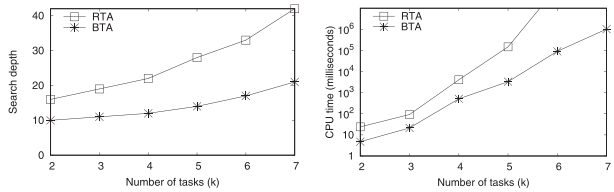
TABLE 5
Experiment Parameters

Parameter	Default value
Number of tasks $\|S\|$	4000
Worker travel distance coefficient tc	0.1
Worker deadline coefficient dc	1.5
Task expiration time coefficient ec	2.5
Maximum acceptable workers $maxW$	1

After constructing the balanced tree by MCS algorithm (in Section 3.2.2) and RTC algorithm (in Section 3.2.3) based on the new worker dependency relationship, the search algorithm has to be modified to achieve the goal of maximizing the overall task assignments when dealing with the redundant task assignment problem. Algorithm 4 depicts the improved DFSearch process. Once the search algorithm assigns each maximal valid task set for each available worker in W_N (line 9 and 11), it finds the task set Q'' from the current maximal valid task set Q, in which the tasks have already used their capacity, i.e., for any task s in Q'', it has already been assigned to $maxW_s$ workers (line 14-18). Then the algorithm recursively calls the $Improved\_DFSearch$ procedure by passing in the updated remaining task set $(S - Q'')$, updated worker set $(W_N - w)$ and the updated heuristic $(h - |Q'|)$ (line 20).

Algorithm 4. Improved_DFSearch

```
    Input: N, S, W_N, h
    Output: Opt
1   Opt ← 0;
2   S.n ← 0;
3   t ← 0;
4   ac ← +∞;
5   UB(N) ← compute the upper bound of assigned tasks for
    the sub-tree rooted with N;
6   if UB(N) < h then
7     return 0;
8   if W_N ≠ ∅ then
9     for each worker w ∈ W_N do
10      Q_w = Q_w − Q';
11      for each maximal valid task set Q ∈ Q_w do
12        Q' ← Q' ∪ Q;
13        c(l_w, d_w, Q) ← compute the travel cost from l_w to d_w
          passing Q with the corresponding valid route;
14        Q'' ← ∅;
15        for each task s ∈ Q do
16          s.n+ ← 1;
17          if s.n = maxW_s then
18            Q''+ ← s;
19        t+ ← c(l_w, d_w, Q);
20        Opt ← max{Improved_DFSearch(N, S − Q'', W_N −
          w, h − |Q|) + |Q|, Opt};
21        h ← Opt;
22        ac ← min{t, ac};
23        Q' ← ∅;
24  else
25    for each child node N_i of N do
26      Opt+ ← Improved_DFSearch(N_i, S, W_{N_i}, LB(N_i));
27  return Opt;
```



(a) Search depth (b) CPU time

Fig. 3. Performance of worker partition: Effect of $|S|$.

5 Experiment

5.1 Experiment Setup

Due to the lack of benchmark for spatial crowdsourcing algorithms, we use a real trajectory dataset generated by taxis in a big city to simulate the travel behaviors of workers, in which the lengths (i.e., travel distances) of these trajectories vary from 1 km to 30 km and the travel times vary from $105s$ to $2723s$. The average speed of each worker can be easily computed based on the travel distance and time of each trajectory. For each test we randomly choose 1000 trajectories from the dataset with similar (Euclidean) travel distance controlled by travel distance coefficient tc, which is defined as the ratio between the origin-destination distance and the maximum travel distance in the dataset. Each worker's deadline is set by multiplying her actual travel time with a deadline coefficient dc. Then we uniformly generate $|S|/|W|$ tasks inside the workers' elliptic reachable region and set each task's expiration time as $ec * c(l_w, l_s)/speed_w$, where $c(l_w, l_s)$ is the travel distance from origin to the location of task and $speed_w$ is the worker's average speed. The default values of all the parameters used in our experiments are summarized in Table 5. For each experiment, we run 50 test cases and report the average results. All the algorithms are implemented on an Intel Core i5-2400 CPU @ 3.10G HZ with 8 GB RAM.

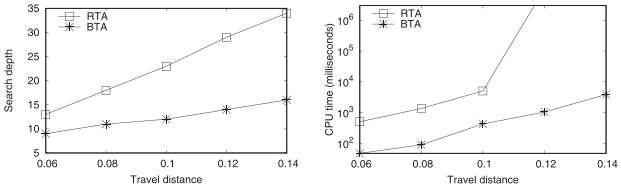
5.2 Experiment Results

5.2.1 Performance of Worker Partition

In this part we evaluate the performance of worker partition phase and its impact to subsequent search. While applying the same graph partitioning algorithm (Section 3.2.2), we introduce a baseline algorithm for tree construction, Random Tree Construction algorithm (RTA), which randomly selects a worker cluster as the root node of sub-tree. Two metrics are compared between RTA and our proposed balanced tree-construction algorithm (BTA): 1) search depth: the maximum number of workers enumerated when searching from the root node to leaf nodes within one depth-first traversal; 2) CPU time: the CPU time cost for finding the optimal assignment with the resulting tree.

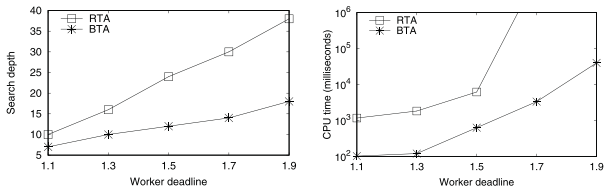
Effect of $|S|$. First, we investigate how the number of tasks affects the resulting trees. As shown in Fig. 3a, though the search depths of both tree construction algorithms increase with $|S|$, BTA can generate a much more balanced tree, which in turn leads to more efficient search than RTA as confirmed in Fig. 3b.

Effect of tc. As illustrated in Fig. 4, the performances of both algorithms deteriorate as the worker travel distance coefficient increases. This is because the dependency among workers increases when there are more reachable tasks for



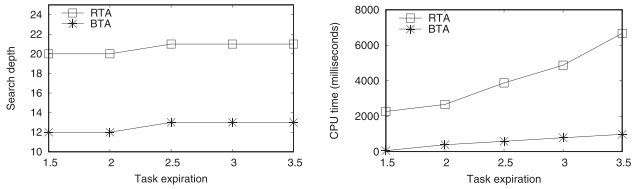
(a) Search depth (b) CPU time

Fig. 4. Performance of worker partition: Effect of tc.



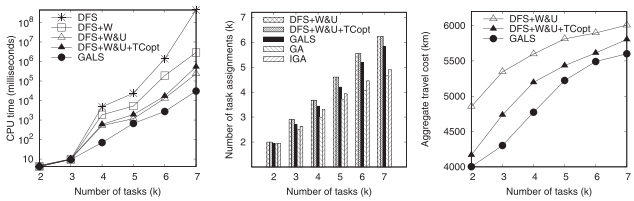
(a) Search depth (b) CPU time

Fig. 5. Performance of worker partition: Effect of dc.



(a) Search depth (b) CPU time

Fig. 6. Performance of worker partition: Effect of ec.



(a) CPU time (b) Task assignment (c) Travel cost

Fig. 7. Performance of search: Effect of $|S|$.

each worker. Another observation is that, the performance gap of both approaches in terms of search cost is also increasing. This is due to the fact, when the tree is unbalanced, the search cost is more sensitive to the average number of valid task sets of each worker that increases with tc. In such circumstances, the benefits of a more balanced tree become more significant.

Effect of dc. Evidently, the effect of workers' travel distance coefficient tc and deadline coefficient dc are strongly correlated, which explains why the impact of dc shares the similar trend with that of tc (see Fig. 5a). While the search cost of both algorithms increases quickly with dc, RTA deteriorates much faster and cannot even return a result within tolerated time when $dc > 1.6$, which demonstrates the importance of tree structure to the search performance.

Effect of ec. As shown in Fig. 6a, the search depth is not affected by the expiration time. This is because the dependency among workers does not change much as long as the reachable task set remains stable for each worker. However, as noted in Fig. 6b, the search cost of RTA increases much faster than BTA since a greater ec results in more valid task sets hence more VTS enumeration during the search. This again confirms the superiority of a tree with more balanced sub-trees.

5.2.2 Performance of Task Assignment Algorithm

In this part, we compare the efficiency (i.e., CPU time) of following algorithms:

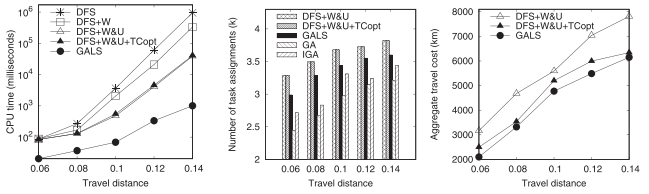
1) DFS: our proposed $DFSearch$ algorithm based on the balanced tree.
2) $DFS + W$: DFS with optimization of re-ordering sub-tree traversal based on the number of workers.
3) $DFS + W\&U$: $DFS + W$ with optimization of UB computation.
4) $DFS + W\&U + TCopt$: $DFS + W\&U$ with travel cost optimization.
5) $GALS$: Global Assignment and Local Scheduling algorithm that iteratively assigns tasks for workers based on maximum flow method and schedules the suitable tasks for each worker [8], where the capacity of worker w is set to the number (i.e., $|RS_w|$) of her reachable tasks. When scheduling tasks, GALS has to ensure the worker can arrive her destination before deadline after completing all the scheduled tasks.

For effectiveness of task assignment, we first compare the number of task assignments in the following methods:

1) $DFS + W\&U$.
2) $DFS + W\&U + TCopt$.
3) $GALS$.
4) GA: Greedy Algorithm that assigns each worker with the maximal valid task sets from the unassigned tasks, until all the tasks are assigned or all the workers are exhausted.
5) IGA: Iteratively Greedy Algorithm that repeats GA procedure multiple times with every worker as the first one to be assigned and choose the best assignment as the final result.

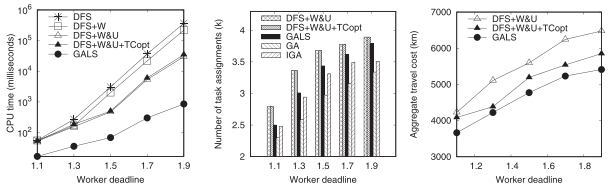
Moreover, we compare the travel cost among $DFS + W\&U$, $DFS + W\&U + TCopt$ and $GALS$ algorithms. The effectiveness is measured as the aggregate travel cost, which is the sum of the Euclidean distances of the valid routes for all workers while satisfying the spatio-temporal constraints of workers and tasks.

Effect of $|S|$. In this set of experiments, we evaluate the scalability of all the approaches by varying the number $|S|$ of tasks from 2k to 7k. As we can see from Fig. 7a, all our proposed algorithms have similar performance when the number of tasks is low, which means there are not many benefits gained from the optimizations. However, the benefits of reordering sub-tree traversal and tighter upper bound become more obvious when $|S| > 3000$. Another observation is that, the CPU cost of $DFS + W\&U + TCopt$ is a little bit higher than that of $DFS + W\&U$, but it saves huge travel cost (see Fig. 7c), which demonstrates the benefits of our proposed travel cost optimization strategy. Although $GALS$ is fastest among all the methods, it assigns less tasks than our proposed methods (i.e., $DFS + W\&U$ and $DFS + W\&U + TCopt$), shown in Fig. 7b. Fig. 7b also depicts that the greedy algorithms are more disadvantaged than others with the



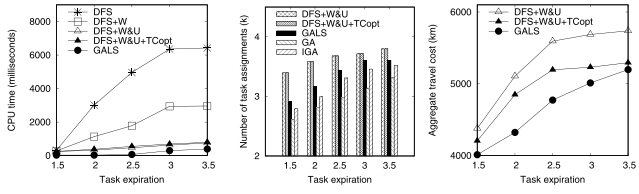
(a) CPU time (b) Task assignment (c) Travel cost

Fig. 8. Performance of search: Effect of tc.



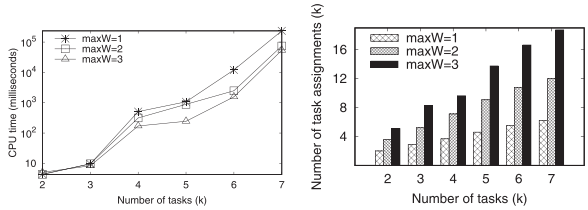
(a) CPU time (b) Task assignment (c) Travel cost

Fig. 9. Performance of search: Effect of dc.



(a) CPU time (b) Task assignment (c) Travel cost

Fig. 10. Performance of search: Effect of ec.



(a) CPU time (b) Task assignment

Fig. 11. Performance of redundant task assignment: Effect of $|S|$.

growth of task numbers. From Fig. 7c we can see $GALS$ obtains lower travel cost than both $DFS + W\&U$ and $DFS + W\&U + TCopt$. This is because, $DFS + W\&U$ and $DFS + W\&U + TCopt$ have more assigned tasks and workers need to take more travel cost to perform such assigned tasks.

Effect of tc. Fig. 8 illustrates the effect of tc on the performance of all algorithms. As expected in Fig. 8a, increasing the travel distance will incur more CPU time for all algorithms. This may be due to the fact that, more valid tasks need to be searched when worker's travel distance is getting longer. With the increase of travel distance, search space increases explosively that explains the more benefits of our optimized approaches. Fig. 8b shows that the optimality of greedy algorithms deteriorate as each worker has more valid task sets. Furthermore, we notice that $DFS + W\&U + TCopt$ outperforms $DFS + W\&U$ a large margin (up to 24.3 percent) in terms of travel cost, which shows the effectiveness of the proposed travel cost optimization strategy again (see Fig. 8c).

Effect of dc. In this set of experiments, we study the effect of worker's deadline. Not surprisingly, as we can see in Fig. 9a, the performance gaps among the search algorithms (i.e., DFS, $DFS + W$ and $DFS + W\&U$) become larger when the deadlines are more relaxed. Fig. 9b demonstrates that the greedy algorithms' utilization of workers' increased capability to perform more tasks is marginal compared to our exact algorithm. The worker deadline has similar effect on the aggregate travel cost with worker travel distance, as demonstrated in Fig. 9c. The intrinsic reason lies in the more valid task sets generated as the increasing deadline.

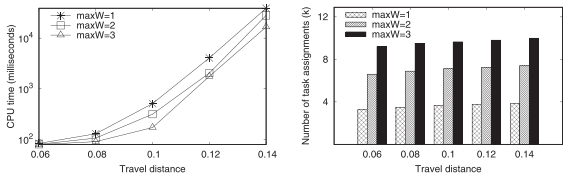
Effect of ec. Fig. 10 illustrates the effect of task expiration time. As expected, longer expiration time means on average each worker has more freedom to schedule the tasks, which results in greater search space. However, this performance deterioration can be greatly relieved by the proposed optimizations. On the other hand, as shown in Fig. 10b, the number of assigned tasks is not heavily affected since the reachable task set of each worker is unchanged. When comes to the travel time, we notice that the increase of aggregate travel cost of our proposed algorithms (i.e., $DFS + W\&U$ and $DFS + W\&U + TCopt$) becomes slower when $ec \geq 2.5$ since with longer task expiration time there is increasing chance that most of the tasks have already been added into the valid task sets and thus few tasks need to be added into the valid task sets.

5.2.3 Performance of Redundant Task Assignment Strategy

Finally, we test the performance of the redundant task assignment strategy proposed in Section 4.2. In particular, we evaluate both efficiency and effectiveness of the three strategies, i.e., a task can only be assigned to 1 worker ($maxW = 1$), a task can be assigned to at most 2 workers ($maxW = 2$) and a task can be assigned to at most 3 workers ($maxW = 3$). Each set of experiments measure the CPU time and the number of task assignments.

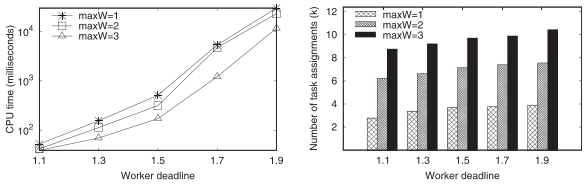
Effect of $|S|$. As demonstrated in Fig. 11a, all the methods become more time consuming when $|S|$ increases since more valid task sets that need to be traversed are generated. It is worth noting that the strategy of $maxW = 1$ is most time consuming. This is because that workers tend to be dependent with fewer other workers when the maximum number of acceptable workers for tasks gets larger, which generates a simpler tree and thus makes the search procedure simpler and more effectual. Regarding their effectiveness (see Fig. 11b), naturally the number of task assignments generated from all strategies increases when more tasks are involved. In addition, the strategy of $maxW = 3$ performs the best followed by that of $maxW = 2$ and then $maxW = 1$. It is interesting to see that at least 71.4 percent tasks can be assigned to 2 workers when $maxW = 2$ and at least 41.3 percent tasks can be assigned to 3 workers when $maxW = 3$, which can improve the accuracy of task results.

Effect of tc. Fig. 12 shows the performance of travel cost optimization by changing over the length of workers' travel distance. A longer travel distance is more likely to make more tasks valid for a worker, which takes more search time and provides more task assignments, which is indicated in Figs. 12a and 12b.



(a) CPU time (b) Task assignment

Fig. 12. Performance of redundant task assignment: Effect of tc.



(a) CPU time (b) Task assignment

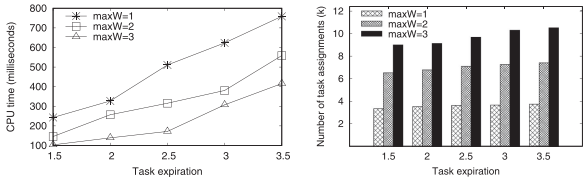
Fig. 13. Performance of redundant task assignment: Effect of dc.

Effect of dc. We also study the effects of the worker deadline coefficient by varying it from 1.1 to 1.9. As expressed in Fig. 13a, all the methods have the trend of a growth in both the CPU time and the number of task assignments, whose reason is similar with the effect of worker travel cost, i.e., as dc gets larger, the search procedure needs to check and assign more valid tasks for workers.

Effect of ec. In this set of experiment, we change the task expiration time coefficient ec from 1.5 to 3.5. Obviously, as illustrated in Fig. 14a, the CPU time of all the methods are growing when ec is enlarged. Another observation is that the strategy is most efficient when $maxW = 3$, which is approximately 33 percent slower than that of $maxW = 2$ and 55 percent slower than that of $maxW = 1$. The reason behind it is that the larger $maxW$ in redundant task assignment mode leads to a simpler tree to be searched with simpler dependence among workers. In terms of the number of task assignments, more task assignments are generated as $maxW$ grows, which is clearly demonstrated in Fig. 14b.

6 Related Work

Spatial Crowdsourcing (SC) is a new class of crowdsourcing, which employs smart device carriers as workers to physically move to some specified locations and perform spatial tasks [11], [18]. Based on the task publish mode, SC can be classified into *Server Assigned Tasks (SAT) mode* and *Worker Selected Tasks (WST) mode* [13]. In SAT mode, the server assigns each task to nearby workers based on the system optimization goals such as maximizing the number of assigned tasks after collecting all the locations of workers [4], [8], [13], [14], [23], [25], maximizing the total payoff from assigned tasks [2], maximizing the expected total utility achieved by all workers [3], [21], [24], maximizing task reliability for dynamic task assignment [10], maximizing the expected quality of results from workers by a real-time budget-aware task package allocation [26], or maximizing the spatial/tempral coverage where/when workers perform tasks [11]. For instance, [8] considers task assignment and scheduling at the same time, in which an approximate approach is developed that iteratively improves the



(a) CPU time (b) Task assignment

Fig. 14. Performance of redundant task assignment: Effect of ec.

assignment and scheduling to achieve more completed tasks. However, their paper assumes that each worker can only perform tasks in a specific spatial region, so the search space in their problem settings is much smaller than ours. Moreover, their work proposes an approximate algorithm while we offer an exact solution. In WST mode, the server publishes various spatial tasks online, and workers can select any tasks without the coordination with the server [7]. For example, Deng et al. [7] formulate SC as a scheduling problem by reducing it into a specialized *Traveling Salesman Problem*. The exact and approximation algorithms are proposed to find a schedule to maximize the number of tasks that can be completed by a worker when both travel cost of workers and expiration time of tasks are taken into consideration.

Moveover, with spatial crowdsourcing, tasks can be assigned in two different modes: *Single Task Assignment (STA) mode* and *Redundant Task Assignment (RTA) mode* [13]. STA mode assumes that all workers are trusted and can perform the tasks correctly without any malicious intentions, so that each task is only assigned to one worker in STA mode. However, there inevitably exist some malicious workers that might intentionally complete tasks incorrectly (i.e., being dishonest about physically moving to the locations of tasks). Therefore, RTA mode is proposed to improve the validity of task completion by assigning each task to several nearby workers. In RTA mode, the task completion result with the majority vote is regarded as correct.

Among the above studies in SC, travel cost plays a crucial role, due to the fact that SC workers have to physically move to the locations of spatial tasks in order to perform them [9], [15], [22]. For instance, considering *task localness*, which refers to workers' preferences based on their travel cost (i.e., workers are more likely to accept nearby tasks), [9] proposes an effective task assignment framework by modelling task acceptance rate as a decreasing function of travel distance. Cheung et al. [15] formulate the interactions among users as a non-cooperative Task Selection Game (TSG), and propose an Asynchronous and Distributed Task Selection (ADTS) algorithm, which balances the rewards and travel costs of the workers for completing tasks.

7 Conclusion

In this paper we study the problem of finding the optimal task assignment for destination-aware spatial crowdsourcing, where each worker can complete all the assigned tasks before their expiration time and reach her destination before a given deadline. To settle the intractable complexity of this problem, we propose a graph partitioning based approach to decompose the complex worker dependency graph into smaller

independent worker clusters, and a tree construction algorithm to organize the clusters into balanced tree structure. Then a depth-first search strategy is devised with effective lower and upper bounds to avoid unpromising traversals. Finally, we further optimizing the original algorithm by proposing strategies to reduce the overall travel cost and allow each task to be assigned to multiple workers in order to generalize the applicability of the proposed framework. Extensive empirical study demonstrates our proposed solution is efficient enough to deliver the maximum number of assignment within reasonably small amount of time, and the optimization strategies can also result in less overall travel cost and increased total task assignment when redundant task assignment is requested.

Acknowledgments

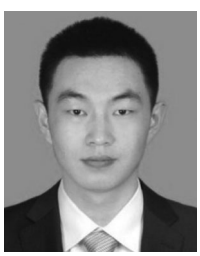
This work is partially supported by the Natural Science Foundation of China (No. 61532018, 61836007, and 61832017), the Australian Research Council (Grants No. DP170101172), and the Major Project of Zhejiang Lab (No. 2019DH0ZX01).

References

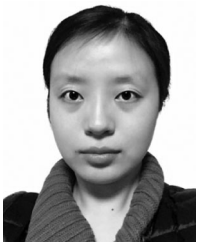
[1] J. Blair and B. Peyton, "An introduction to chordal graphs and clique trees," in *Proc. Graph Theory Sparse Matrix Comput.*, 1993, pp. 1–29.
[2] C. Chen, S. F. Cheng, A. Gunawan, A. Misra, K. Dasgupta, and D. Chander, "Traccs: A framework for trajectory-aware coordinated urban crowd-sourcing," in *Proc. AAAI Conf. Human Comput. Crowdsourcing*, 2014, pp. 30–40.
[3] C. Chen, S. F. Cheng, H. C. Lau, and A. Misra, "Towards city-scale mobile crowdsourcing: Task recommendations under trajectory uncertainties," in *Proc. 24th Int. Conf. Artif. Intell.*, 2015, pp. 1113–1119.
[4] P. Cheng, X. Lian, L. Chen, and C. Shahabi, "Prediction-based task assignment in spatial crowdsourcing," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 997–1008.
[5] P. Cheng, X. Lian, Z. Chen, R. Fu, L. Chen, J. Han, and J. Zhao, "Reliable diversity-based spatial crowdsourcing by moving workers," *VLDB Endowment*, vol. 8, no. 10, pp. 1022–1033, 2015.
[6] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos, "Anonysense: Privacy-aware people-centric sensing," in *Proc. 6th Int. Conf. Mobile Syst. Appl. Services*, 2008, pp. 211–224.
[7] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *Proc. 21st ACM SIGSPATIAL Int. Conf. Advances Geographic Inf. Syst.*, 2013, pp. 324–333.
[8] D. Deng, C. Shahabi, and L. Zhu, "Task matching and scheduling for multiple workers in spatial crowdsourcing," in *Proc. 23rd SIGSPATIAL Int. Conf. Advances Geographic Inf. Syst.*, 2015, Art. no. 21.
[9] G. Ghinita, G. Ghinita, and C. Shahabi, "A framework for protecting worker location privacy in spatial crowdsourcing," *VLDB Endowment*, vol. 7, pp. 919–930, 2014.
[10] U. U. Hassan and E. Curry, "Efficient task assignment for spatial crowdsourcing: A combinatorial fractional optimization approach with semi-bandit learning," *Expert Syst. Appl.*, vol. 58. pp. 36–56, 2016.
[11] Z. He, J. Cao, and X. Liu, "High quality participant recruitment in vehicle-based crowdsourcing using predictable mobility," in *Proc. Conf. Comput. Commun.*, 2015, pp. 2542–2550.
[12] L. Kazemi and C. Shahabi, "A privacy-aware framework for participatory sensing," *SIGKDD Explorations Newslett.*, vol. 13, no. 1, pp. 43–51, 2011.
[13] L. Kazemi and C. Shahabi, "Geocrowd: Enabling query answering with spatial crowdsourcing," in *Proc. 20th Int. Conf. Advances Geographic Inf. Syst.*, 2012, pp. 189–198.
[14] L. Kazemi, C. Shahabi, and L. Chen, "Geotrucrowd: Trustworthy query answering with spatial crowdsourcing," in *Proc. 21st ACM SIGSPATIAL Int. Conf. Advances Geographic Inf. Syst.*, 2013, pp. 314–323.
[15] H. C. Man, R. Southwell, F. Hou, and J. Huang, "Distributed time-sensitive task selection in mobile crowdsensing," in *Proc. 16th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2015, pp. 157–166.
[16] N. Robertson and P. Seymour, "Graph minors. II. algorithmic aspects of tree-width," *J. Algorithms*, vol. 7, no. 3, pp. 309–322, 1986.
[17] D. Rose, "Triangulated graphs and the elimination process," *J. Math. Anal. Appl.*, vol. 32, no. 3, pp. 597–609, 1970.
[18] T. Song, Y. Tong, L. Wang, J. She, B. Yao, L. Chen, and K. Xu, "Trichromatic online matching in real-time spatial crowdsourcing," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 1009–1020.
[19] J. Surowiecki, "The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations," *Personnel Psychology*, vol. 59, no. 4, pp. 982–985, 2006.
[20] R. Tarjan and M. Yannakakis, "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs," *SIAM J. Comput.*, vol. 13, no. 3, pp. 566–579, 1984.
[21] Y. Tong, L. Chen, Z. Zhou, H. V. Jagadish, L. Shou, and W. Lv, "SLADE: A smart large-scale task decomposer in crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 8, pp. 1588–1601, Aug. 2018.
[22] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis," *VLDB Endowment*, vol. 9, no. 12, pp. 1053–1064, 2016.
[23] Y. Tong, J. She, B. Ding, and L. Wang, "Online mobile micro-task allocation in spatial crowdsourcing," in *Proc. IEEE 32nd Int. Conf. Data Eng.*, 2016, pp. 49–60.
[24] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye, "Dynamic pricing in spatial crowdsourcing: A matching-based approach," in *Proc. Int. Con. Manage. Data*, 2018, pp. 773–788.
[25] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu, "Flexible online task assignment in real-time spatial data," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1334–1345, 2017.
[26] P. Wu, E. W. Ngai, and Y. Wu, "Toward a real-time and budget-aware task package allocation in spatial crowdsourcing," *Decision Support Syst.*, vol. 110, 107–117, 2018.
[27] Y. Zhao, Y. Li, Y. Wang, H. Su, and K. Zheng, "Destination-aware task assignment in spatial crowdsourcing," in *Proc. ACM Conf. Inf. Knowl. Manage.*, pp. 297–306, 2017.

Yan Zhao received the master's degree in geographic information system from the University of Chinese Academy of Sciences, in 2015. She is currently working toward the PhD degree in Soochow University. Her research interests include spatial database and trajectory computing.

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



Yang Li received the bachelor's degree in computer science and technology at Soochow University, in 2015. He is currently working toward the master's degree at Soochow University. His research interests include data mining and spatial crowdsourcing.



Han Su received the BS degree in software engineering from Nanjing University, in 2011, and the PhD degree in computer science from the University of Queensland, in 2015. She is currently an associate professor with the Big Data Research Center, University of Electronic Science and Technology of China. Her research interests include trajectory querying and mining.

Jiajun Liu received the BEng degree from Nanjing University, China, in 2006, and the PhD degree from the University of Queensland, Australia, in 2012. He is an associate professor at Renmin University of China. Before joining Renmin University he was a postdoctoral fellow with the CSIRO of Australia from 2012 to 2015. From 2006 to 2008, he also worked as a researcher/software engineer for the IBM China Research/Development Labs. His main research interests include in multimedia and spatio-temporal data management and mining. He serves as a reviewer for multiple journals such as the *VLDB Journal*, the *IEEE Transactions on Knowledge and Data Engineering*, the *IEEE Transactions on Multimedia*, and as a PC member for ACM MM and CCF Big Data.

Xiaofang Zhou received the bachelor's and master's degrees in computer science from Nanjing University, in 1984 and 1987, respectively, and the PhD degree in computer science from the University of Queensland, in 1994. He is a professor of computer science with the University of Queensland. He is the head of the Data and Knowledge Engineering Research Division, School of Information Technology and Electrical Engineering. He is also a specially appointed adjunct professor with Soochow University, China. His research is focused on finding effective and efficient solutions to managing integrating, and analyzing very large amounts of complex data for business and scientific applications. His research interests include spatial and multimedia databases, high performance query processing, web information systems, data mining, and data quality management. He is a fellow of the IEEE.