Coverage and Workload Cost Balancing in Spatial Crowdsourcing

Ning Wang, Jie Wu and Pouya Ostovari
Department of Computer and Information Sciences, Temple University, USA
Email: {ning.wang, jiewu, ostovari}@temple.edu

***Abstract*—This paper addresses the coverage and workload-balancing requirements of worker recruiting in spatial crowdsourcing. That is, the recruited workers should be able to visit all the crowdsourcing locations to satisfy a certain quality, e.g., traffic monitoring or climate forecast. In addition, each crowdsourcing operation has a cost, e.g., data traffic or energy consumption, and each crowdsourcing location might have a crowdsourcing budget for the visited workers. The objective of this paper is to find a worker recruiting algorithm, which ensures the coverage requirement and minimizes the maximal crowdsourcing cost for any crowdsourcing location. We gradually discuss the problem from the 1-D scenario to the general 2-D scenario. In the 1-D scenario, we propose a bounded directional greedy algorithm first. Then, we propose a PTAS extension. A dynamic programming solution is further proposed with a higher computation complexity. In the 2-D scenario, we propose a randomized rounding algorithm with an $O(\frac{\log n}{\log \log n})$ approximation ratio in a high probability. Extensive experiments on realistic traces demonstrate the effectiveness of the proposed algorithms.**

***Index Terms*—Crowdsourcing, crowdsensing, task allocation.**

I. Introduction

Spatial crowdsourcing [1], also called participatory crowdsourcing, has emerged in the past few years with the ubiquity of mobile devices and vehicles equipped with high-fidelity sensors and the development of wireless networks (e.g., WiFi and LTE). It aims at exploiting mobile users to actively collect and report data by using their mobile devices for a given campaign. The unique challenge of spatial crowdsourcing is that it consists of location-specific tasks. The people who agree to participate in the spatial crowdsourcing, called workers, have to physically be at specific locations to complete the tasks. There are many applications, such as geographical data generation (e.g., OpenStreetMap [2]) and road traffic monitoring (e.g., Waze [3]).

Existing spatial crowdsourcing mechanisms do not consider the coverage requirement [4–7]. However, the crowdsourcing coverage is the requirement of many practical applications, such as weather forecasts, e.g., DroneSense [8], and traffic route optimization, e.g., CalTel [9], Nericell [10], and GreenGPS [11]. It is because missing data from any crowdsourcing location might have a huge impact on the final result, therefore, the desired performance (e.g., QoS) cannot be guaranteed without the coverage constraint. Another problem, which is important but often ignored in spatial crowdsourcing, is the workload balancing requirement for each crowdsourcing location. In reality, there is a cost (e.g., battery consumption, time, cellular traffic, and pay-off) for each crowdsourcing

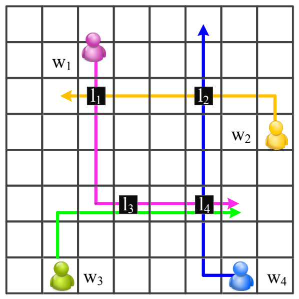


Fig. 1. An illustration of crowdsourcing model in this paper, where the black blocks are spatial crowdsourcing tasks.

operation, and the budget in each crowdsourcing location is limited. For example, workers collect environment/traffic information from sensors in a target area, the solar-charging sensor in each location has limited battery per day, so the communication energy consumption cannot exceed the sensor's battery. Another application scenario example is that there exists a roadside unit in each crowdsourcing location and it has a limited data traffic budget through cellular networks.

In this paper, we address the aforementioned challenges in the spatial crowdsourcing. In the model of this paper, there are a set of spatial crowdsourcing locations and workers. The trajectory of each worker is known, and there is a crowdsourcing cost when it visits a crowdsourcing location. Our goal is to find a recruitment solution which ensures all the crowdsourcing locations can be visited by at least one worker, while minimizing the maximal cost for all crowdsourcing locations. We refer to this problem as the Coverage and Balanced Crowdsourcing Recruiting (CBCR) problem.

To illustrate the CBCR problem, there is a motivation example in Fig. 1, where there are 4 workers, w_1, w_2, w_3, and w_4 with 4 crowdsourcing locations, namely, l_1, l_2, l_3, and l_4. Assume workers have an identical cost for visiting any crowdsourcing location in this toy example. In Fig. 1, the worker w_1 visits l_1, l_3, and l_4. We cannot only select w_1, since this recruiting solution does not satisfy the coverage constraint. To satisfy the coverage constraint, there are three feasible recruitment strategies, i.e., $\{w_1, w_2\}$, $\{w_1, w_4\}$, and $\{w_2, w_3\}$. For the first two solutions, the crowdsourcing location l_1 or l_4 is covered twice. However, in the last solution, all the crowdsourcing locations are only covered once, which is better in terms of balancing the workload.

The proposed CBCR problem is proven to be NP-hard in the general case. Then, we discuss the solution of the CBCR problem in the 1-D scenario. In the 1-D scenario, we propose to cover crowdsourcing locations directionally with a performance bound. After that, a PTAS extension is introduced in this scenario to trade-off the computation complexity and the performance. Lastly, the dynamic programming approach is proposed to find the optimal solution in the 1-D scenario. In the general 2-D scenario, we propose a randomized rounding algorithm with an expectation bound.

The contributions of this paper are summarized as follows:

- To our best knowledge, we are the first to consider the coverage requirement and the workload balancing of the crowdsourcing locations in the spatial crowdsourcing.
- In the 1-D scenario, we propose a PTAS solution. Later, we propose a dynamic programming approach to find the optimal solution.
- In the general 2-D scenario, we propose a randomized rounding algorithm which can solve the program effectively with a high probability.

The remainder of the paper is organized as follows. The related works are in Section II. The problem statement is introduced in Section III. The solution in the 1-D scenario is provided in Section IV. The solution for the 2-D scenario is presented in Section V. The experimental results are shown in Section VI. We conclude the paper in Section VII.

II. Related Works

With the wide adaptation of spatial crowdsourcing applications, task coverage and participant selection in the spatial crowdsourcing system have drawn much attention from researchers in recent years [4–7, 12–14]. The existing works can be mainly categorized into two types:

Worker trajectory planning: In this type of model, the worker's trajectory can be controlled or planned by the server [6, 7, 15]. Previous researchers [16] have produced many works where there is only one worker in their spatial crowdsourcing model. In [6], Each task has a deadline, a feasible route of a worker should make sure that all the tasks in the route can be finished before their deadline. They proposed an approach to maximize the number of tasks that a worker can finish. In [4], they proposed a model with multiple workers. However, their approach is an iteration solution, which considers workers one-by-one. Therefore, they did not really address the multiple worker collaborative crowdsourcing. That is also the reason why the method in [4] does not have a performance bound. In [7] , authors considered the event conflict for a worker, then provided a mapping solution with pruning to reduce the complexity.

Trajectory coverage: In this type of model, the worker's trajectory is pre-determined [12, 17]. We argue that in many crowdsourcing applications, the worker's trajectory is determined, e.g., Waze [3]. There are many theoretical studies on task assignments and participant selection problems, playing trade-offs among the crowdsourcing cost and coverage range [12, 17]. The difference between their models and our model in this paper is that they consider maximizing the coverage range. However, in many scenarios, ensuring the coverage in a certain area, such as traffic monitoring or surveillance, is very important. In [13], authors considered the coverage requirement in the crowdsourcing, and minimized the overall recruiting cost. The difference between their work and this paper is that they consider the overall crowdsourcing cost. We argue that the workload balance is very important, thus, we consider the crowdsourcing cost for each crowdsourcing location. In [18], the authors considered the workload constraint but the coverage constraint was not considered.

III. Model and Problem

In this section, we introduce the network model used in this paper, followed by the problem formulation. The hardness of the proposed problem is shown at the end.

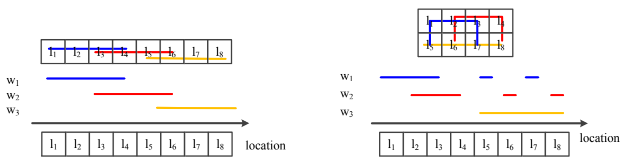
A. Model

In this paper, we discuss a spatial crowdsourcing scenario under a centralized matter. We assume that the spatial crowdsourcing area can be mapped into a 2-D grid topology and each crowdsourcing location belongs to a grid. There are two types of locations: crowdsourcing location, denoted as a black block in Fig. 1 and regular location. In a crowdsourcing location, there is a crowdsourcing task. In a regular location, there is not a crowdsourcing task. Suppose there are m crowdsourcing locations, $L = \{l_1, l_2, \cdots, l_m\}$.

In the network there are N workers (e.g., people or vehicles), $W = \{w_1, w_2, \cdots, w_n\}$, who agreed to accept the crowdsourcing task. Each worker has a known crowdsourcing trajectory, t_i, which is pre-determined in this paper. That is, we have a vector T, $T = \{t_1, t_2, \cdots, t_n\}$ for the N worker. The length of t_i is denoted as $|t_i|$, which is the number of crowdsourcing locations that w_i passes. Workers can only move in four directions at most, up, down, left, and right to mimic the way that people move along roads in reality. An illustration of the network model is shown in Fig. 1. Once a worker reaches a crowdsourcing location, the task in that location is considered to be covered, and there is a corresponding crowdsourcing cost, c_i, e.g., payoff, energy or data traffic cost. Therefore, we have a cost vector C, $C = \{c_1, c_2, \cdots, c_n\}$, A recruitment policy is a vector $X = \{x_1, x_2, \cdots, x_n\}$, which determines whether worker w_i is selected ($x_i = 1$) or not ($x_i = 0$) to conduct the crowdsourcing task. The workload of a crowdsourcing location is denoted as $\sum c_i x_i$, once t_i includes this location. The application for this model is like Waze [3], e.g., the drivers share the traffic information on their way home.

B. Problem Formulation

To address the coverage requirement of spatial crowdsourcing, e.g., traffic monitoring, route recommendation, climate forecast, and surveillance systems, we argue that the data should be collected from all $|L|$ crowdsourcing locations before calculation to ensure a certain reliability level. In addition, a practical issue in crowdsourcing is that there is a



(a) 1-D scenario (b) 2-D scenario

Fig. 2. The trajectory of workers in 1-D and 2-D scenarios.

crowdsourcing cost to pay the visited workers and there should be a budget for each crowdsourcing location. Therefore, we propose the Coverage and Balanced Crowdsourcing Recruiting (CBCR) problem in this paper, which is formulated as follows.

$$\begin{aligned} \min \quad & \max_{i} \sum_{l_i \in t_j} c_j x_j \\ \text{s.t.} \quad & \sum_{l_i \in t_j} x_j \geq 1, \quad \forall l_i \qquad x_j \in \{0,1\}, \end{aligned} \tag{1}$$

where the objective is to find a worker recruitment solution so that the maximum crowdsourcing budget for all crowdsourcing locations is minimized, and the constraint ensures that every crowdsourcing location is covered.

C. Hardness of CBCR problem

Theorem 1. *The proposed CBCR problem is NP-hard.*

Proof. To show the decision version of the CBCR (CBCR-D) $\in$ NP, suppose a recruiting policy X is given. Clearly, we can verify the correctness of X in polynomial time. Specifically, the complexity of the verification algorithm is $O(m)$, and the number of crowdsourcing location is m in X.

To show that CBCR-D $\in$ NP-hard, we reduce the tripartite matching problem to it in polynomial time, which is NP-complete [19]. The tripartite matching problem can be formulated as: given sets B, G, and H, each containing n elements and a ternary relation $T \in B \times G \times H$, find a set of n triples in T, no two of which have a component in common. We present a polynomial time reduction and construct an instance of CBCR-D as follows:

($\leftarrow$) Consider an instance of CBCR-D in which $c_i = 1$, $|U| = m$, $|S| = n$, and $D = 1$. Each trajectory visits exactly 3 crowdsourcing locations. Three crowdsourcing locations can be partitioned into three equal sets B, G, and H where each set in T contains one element from each. It is equivalent to that in which we build a graph $G = (V, E)$, where V has n nodes, and they are partitioned into three sets, where each of the sets has $n/3$ nodes. Two nodes have an edge if they are visited by one worker. We now argue that the instance (T', T) of tripartite matching is a "yes" instance, i.e. iff there is an instance of CBCR-D.

($\rightarrow$) If the constructed instance of CBCR-D satisfies all the criterion, the selected trajectories form an instance of tripartite matching according to the problem definition. □

Algorithm 1 MG algorithm

Input: The vectors of T and C.

Output: The recruiting vector X.

```
1: while $\exists l \notin \cup t_i$, where $x_i = 1, x_i \in X$ do
2:     Find $x_i$ which will minimize the maximum crowdsourcing cost increase.
3:     Add $x_i$ to $X$
4:     Select the worker whose increase = $temp$
```

Algorithm 2 CO algorithm

Input: The vectors of T and C.

Output: The recruiting vector X.

```
1: Current location, $l_i = l_0$
2: while $l_i < l_m$ do
3:     for $\forall t$, where $l_i \in t$ do
4:         Pick $t_i$ which can increase the coverage most.
5:         Update $l_i$.
```

IV. 1-D Scenario

In this scenario, we assume that all the crowdsourcing locations are in a line-topology. Application for this type of line-topology is road segment monitoring, e.g., a highway situation. On the highway, different vehicles enter in different entrances and leave at different exits. Without loss of generality, let us denote the crowdsourcing locations from one side to the another side as l_1 to l_m for the remainder of this section.

In the 1-D scenario, the overlapping relationship between different trajectories becomes simple. There are two cases in total for two trajectories. (1) They do not have overlapping relationships with each other. (2) They overlap with each other, and all the overlapping locations are contiguous. An illustration of this property in the 1-D scenario is shown in Fig. 2, where there are 8 crowdsourcing locations. In Fig. 2(a), if we map workers' trajectories into a 1-D dimension, their trajectories are contiguous. However, in Fig. 2(b), if we map workers' trajectories into a 1-D dimension, their trajectories might be discontinuous, e.g., t_1's trajectory is $\{l_1, l_2, l_3, l_5, l_7\}$ in Fig. 2(b), which is much more complex.

A. Greedy Approach

In this subsection, we propose two greedy solutions first, followed by the analysis and limitation about the proposed greedy solutions.

1) Min-max greedy algorithm: A natural idea is to select the worker who cannot increase the maximal workload cost much to cover the network in each round. Therefore, we propose the min-max greedy (MG) algorithm as a baseline algorithm in this paper. For the MG algorithm, if the network is not covered, we select the worker who can minimize the maximal workload of all the crowdsourcing locations in the network. The drawback of the MG algorithm is that a single crowdsourcing location might be unnecessarily covered

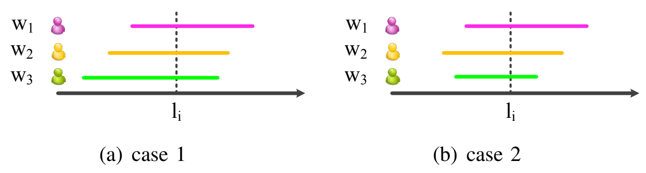


Fig. 3. Two cases for a crowdsourcing location.

multiple times in the MG algorithm and such unnecessary coverage can accumulate and lead to a bad result.

Theorem 2. *In the optimal solution, a crowdsourcing location will not be covered by* 3 *workers in the 1-D scenario.*

Proof. This theorem can be proven by contradiction. If we project all the trajectories for a special crowdsourcing location into the road, we can always find a trajectory whose starting position is the left most, and we can also find a trajectory whose end position is the right most. Then, we can use these two trajectories, possibly as one trajectory, to cover this crowdsourcing location. □

2) Directional coverage algorithm: Facing this drawback of the MG algorithm and theorem 2, we propose the coverage-only (CO) algorithm which covers all the crowdsourcing locations from one side to the other side, i.e., from l_1 to l_m. Each time, we select the worker in an uncovered area, which increases coverage most from left to right. It is easy to prove that the result of the CO algorithm is always feasible which can be proven by contradiction. The CO algorithm takes advantage of the contiguous trajectory overlapping property in the 1-D scenario as shown in theorem 2, and achieves the performance bound, shown in theorem 3.

Theorem 3. *The CO algorithm has a* $2\max_{\forall i,j}\frac{c_i}{c_j}$ *approximation ratio in the 1-D scenario.*

Proof. The proof insight is that each time, we guarantee that a new crowdsourcing location will be covered, so that it avoids unnecessary coverage accumulation. The problem can be proven through contradiction, assuming that there exists a crowdsourcing location, l_i, where 3 workers can visit. Based on the end position of their trajectories from left to right, we denote these three trajectories as t_1, t_2, and t_3 and their staring positions as s_1, s_2, and s_3. Then, for the starting points of t_1 and t_2, there exist two cases: $s_1 < s_2$ or $s_1 > s_2$ as shown in Fig. 3. In the first case, the CO algorithm will select t_1 rather than t_2. In the second case, the CO algorithm will select t_2 instead of t_1. Therefore, neither of these two cases exists in the CO algorithm. Otherwise, there would be a contradiction. As a result, there are at most two workers visiting a crowdsourcing location and the cost is bounded by $2\max_{\forall i,j}\frac{c_i}{c_j}$. □

3) CO-PTAS algorithm: We observe that the bad performance of the CO algorithm is due to the improper recruitment of workers with a large recruiting cost. Therefore, if these jobs are scheduled with a low priority, they cannot have a big influence on the final result. Based on this observation, we propose a Polynomial-Time Approximation Scheme (PTAS) algorithm. Given all the workers, we partition the workers into two sets: costly workers and cheap workers. Let us use the ratio, r_i between the cost of a worker, c_i, and the maximum worker cost. We call a worker, w_i, a costly worker, if $r_i > \varepsilon$. Let B and S denote the set of costly workers and cheap workers respectively, i.e., $B = \{w_i : r_i > \varepsilon/2\}$ and $S = \{w_i : r_i < \varepsilon/2\}$.

Algorithm 3 CO-PTAS algorithm

```
Input: The vector of $T$ and $C$.
 1: Guess the $\varepsilon$ value
 2: for from $t_0$ to $t_n$ do
 3:     if $c_i \le \frac{\varepsilon}{2}\max_{\forall j} c_j$ then
 4:         Add $t_i$ to $S$
 5:     else
 6:         Add $t_i$ to $B$
 7: Call CO algorithm for trajectories in $S$
 8: if the result is a coverage then
 9:     Modify the guess, $\varepsilon$, and repeat
10: else
11:     Return $X$
```

After the partition, we apply the CO algorithm in the set S. Then, the problem becomes to find a feasible solution in set S. Then, according to theorem 2, the optimal value is at most twice as this value. The optimal ε can be found through the binary search. Initially, we find the maximum worker cost in the network. Then, we set $\varepsilon = 1$ and try to check if we can find a feasible schedule through the CO algorithm. If so, we decrease the value of ε and repeat the CO algorithm, otherwise we increase the value of ε.

Theorem 4. *The CO-PTAS algorithm can achieve a* $2 + \varepsilon$ *approximation ratio in the proposed problem.*

The proof is similar to the approximation ratio proof of the CO algorithm. Due to space limitation, we ignore the proof in this paper. For the complexity of the CO-PTAS algorithm, the CO-PATS algorithm at most calls the CO algorithm $\log(n)$ times according to the binary search. Therefore, the overall complexity of the CO-PTAS algorithm is $O(mnlog(n))$.

An illustration of these three algorithms is in Fig. 4, where there are four trajectories, and their costs are $c_1 = 1$, $c_2 = 1.5$, $c_3 = 3$, and $c_4 = 2$, respectively. The MG algorithm will select w_1 first, since it will only increase by 1, followed by w_4, which further increases 1 for the maximum workload cost. Then, to finish the coverage requirement, w_2 should also be selected. As a result, the maximum workload cost is 3.5. As for the CO algorithm, it directionally covers from l_1 to l_4. Therefore, w_1 is selected first, followed by w_3 in the worst case. Then, the maximum workload cost is 3. However, in the CO-PTAS algorithm, when $\varepsilon = 1$, w_3 will have a lower priority for being selected. Then, the CO-PTAS algorithm will use w_1 and w_2 to finish the coverage, and the maximum workload cost is 2.5.

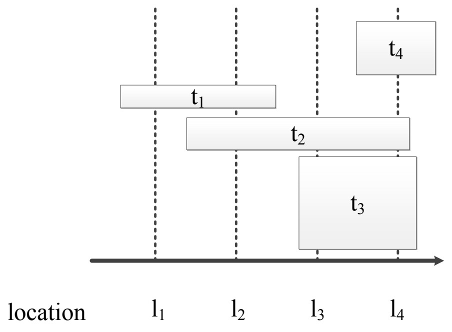


Fig. 4. An example to illustrate the proposed algorithms, where the worker recruiting costs are $c_1 = 1$, $c_2 = 1.5$, $c_3 = 3$, and $c_4 = 2$, respectively.

B. Dynamic Programming Approach

We notice that the trajectory of a worker can only have influence on contiguous locations in the 1-D scenario. Therefore, we can partition the problem into a series of sub-problems to further overcome the improper selection of cost trajectories.

Assume the crowdsourcing locations from one side to another side (e.g., from left to right) is $l_1, l_2, \cdots l_m$. Without loss of generality, we assume trajectories are ordered based on their ending positions. In dynamic programming, we maintain a 2-D vector $d[\cdot,\cdot]$ to store the best result so far. That is, $d[i,j]$ is the optimal solution from crowdsourcing locations l_1 to l_i, and j denotes that the last trajectory t_j which is used to cover location l_i. The objective of dynamic programming is to find $\min d[m,j], \forall j$. Note that if a trajectory t_j does not cover l_i, $d[i,j]$ equals ∞, which ensures that we will not select t_j to cover location l_i. Initially $d[0,j] = 0, \forall j$, which means that before we select any trajectories, the minimal maximum cost is 0. Then we can get the following relationship:

$$d[i,j] = \begin{cases} 0 & \text{i = 0} \\ \underset{i'<i, j'\leq j}{min} \max\{d[i',j'], c_{j'} + c_j\} & \text{Otherwise} \end{cases} \quad (2)$$

where location $l_{i'}$ is any crowdsourcing location that is no smaller than the first crowdsourcing location before the starting point of t_j. j' is any trajectory whose index is smaller than j. For example, if $j = 3$, l'_i means crowdsourcing location l_2, $t_{j'}$ can be t_1 or t_2. The idea behind Eq. 2 is that if we want to find the optimal solutions up to crowdsourcing location l_i with trajectory t_j as the last trajectory, we only need to check all the optimal solution from previous location i' with trajectory $t_{j'}$ to ensure the coverage constraint. If adding t_j to cover the crowdsourcing location from $l_{i'}$ to l_i does not increase the maximum cost, (i.e., $d[i',j'] > c_j + c_{j'}$), we keep the maximum cost, (i.e., $d[i,j] = d[i',j']$). Otherwise, we update the maximum cost of $d[i,j]$, (i.e., $d[i,j] = c_j + c_{j'}$). Note that $c_{j'}$ can be zero, in which case $t_{j'}$ ends at the previous crowdsourcing location before t_j starts. Note that the reason that we can maintain the number of trajectories covering one crowdsourcing location rather than the set of combinations of these trajectories is that the latter is always worse.

An example to illustrate the dynamic programming approach is shown in Fig. 4. For crowdsourcing location l_1, since it is only covered by trajectory t_i, $d[1,1]$ equals 1. Similarly, there are two trajectories covering crowdsourcing location l_2, thus we have $d[2,1] = 1$ and $d[2,2] = 2.5$. The optimal solution that covers the crowdsourcing location from l_1 to l_2 is $\min\{d[2,1], d[2,2]\} = 1$. For crowdsourcing location l_3, there are two trajectories, t_2 and t_3, covering it. For t_2, it is already calculated in the previous crowdsourcing location, therefore, $d[3,2] = d[2,2] = 2.5$. For t_3, it checks all the previous solutions up until crowdsourcing location l_2. Therefore, $d[3,3] = \min\{\max\{d[2,1], c_3\}, \max\{d[2,2], c_2 + c_3\}\} = 3$. In the former case, t_1 ends before t_3, hence $c_{j'} = 0$. Therefore, the optimal solution up to l_3 is $\min\{d[3,2], d[3,3]\} = 2$. For crowdsourcing location t_4, the optimal solution upto l_4 is $\min\{d[4,2], d[4,3], d[4,4]\} = 2.5$.

The complexity analysis of dynamic programming is shown as follows. Dynamic programming requires to store $O(mn)$ states. For each state update, we need to check $O(mn \log n)$ times at most. This is because the algorithm needs to trace back to find the optimal solution in each previous location, $O(m)$, and check the optimal solution at that crowdsourcing location, which is $O(n)$. The dynamic programming approach still needs a sorting algorithm to find the smallest c_j, which is $O(\log n)$. Thus, the overall time complexity is $O(m^2 n^2 \log n)$.

Algorithm 4 Dynamic programming algorithm

```
Input: The vector of $T$ and $C$.
1: Initialize the state record $d[\cdot][\cdot]$
2: for check crowdsourcing location $i$ from $l_1$ to $l_m$ do
3:     for for all crowdsourcing locations $l_{i'}$, $l_{i'} \leq l_i$ do
4:         if $t_j$ can reach location $l_{i'}$ then
5:             $d[i,j] = min_{i'<i,j'\leq j} \max\{d[i',j'], c_{j'} + c_j\}$
6:     Update the $d[i][j]$ for location $l_i$
7: Find min $d[m][\cdot]$
8: Return $X$
```

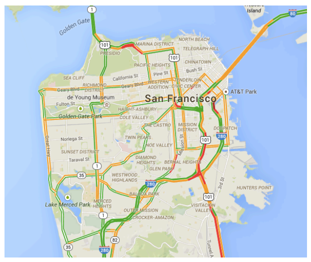
Algorithm 5 Rounding algorithm

```
Input: The vector of $T$ and $C$.
Output: The recruiting vector $X$.
1: Relax the problem into a linear programming formulation.
2: Solve the LP problem and get vector $X^\star$.
3: for for $l_1$ to $l_m$ do
4:     for $\forall l_i \in t_i$ do
5:         Assign each $t_i$ to a interval between 0 and $\sum x_i^\star$.
6:         Randomly generate a value in the whole range.
7:         Pick the $x_i$, if the random value is in its interval.
```

V. General 2-D Scenario

In this section, we discuss approaches to the CBCR problem in the 2-D scenario. Application scenarios for the 2-D scenario is traffic monitoring in an urban area.

In this section, we use a rounding technique [20] to propose a randomized rounding algorithm. For the original problem in Eq. 1, it is equivalent to finding a smallest value θ, which ensures that the workload of any crowdsourcing location is no


(a) San Francisco

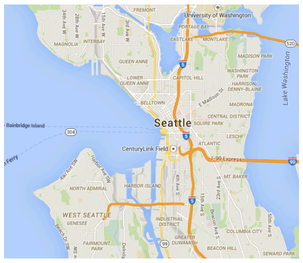

(b) Seattle

Fig. 5. The city map.

larger than θ. Then, if we relax the formulation from $x_i \in \{0,1\}$ to $x_i \in [0,1]$, Eq. 1 becomes a linear programming problem as follows

$$\begin{aligned} \min \quad & \theta \\ \text{s.t.} \quad & \sum_{l_i \in t_j} c_j x_j \le \theta, \quad \sum_{l_i \in t_j} x_i \ge 1, \qquad x_i \in [0,1] \quad \forall i, j. \end{aligned} \tag{3}$$

Eq. 3 can be optimally solved by using the linear programming solver and getting $\theta^\star$ and the corresponding $\{x_1^\star, x_2^\star, \cdots, x_n^\star\}$.

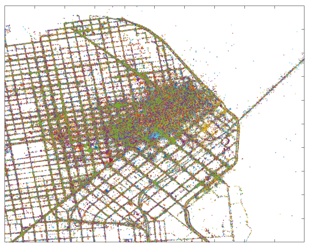
For the original problem, we can use $\{x_1^\star, x_2^\star, \cdots, x_n^\star\}$ to get a randomized rounding result. In detail, the randomized rounding algorithm gives each worker a probability to be recruited. For every crowdsourcing location l_i from l_1 to l_m, we check all trajectories that include l_i. These trajectories are selected based on the their corresponding $x^\star$ in the LP solution. For example, we have three workers with $x_1^\star = 0.5$, $x_2^\star = 0.3$, and $x_3^\star = 0.2$ who visit a crowdsourcing location. The randomized rounding algorithm will randomly generate a number between 0 and 1, the sum of 0.5, 0.3, and 0.2. If the number is between 0 and 0.5, the randomized rounding algorithm will pick the first worker. If the number is between 0.5 and 0.8, pick the second worker, and if the number is between 0.8 and 1, pick the third worker.

The result generated by the randomized rounding algorithm is always a feasible solution. This is because the random assignment ensures that one trajectory for a crowdsourcing location will be selected. The performance of the randomized rounding algorithm is as follows. Since all workers are selected with probability x_i independently, we get the

$$Pr[\sum_{T:l_i \in t_j} c_j E[x_j]] = \sum_{T:l_i \in t_j} c_j E[x_j] = \sum c_j \cdot x_j^\star \le \theta^\star, \tag{4}$$

the expected cost on any location is at most $\theta^\star$. However, since we have many crowdsourcing locations in the network, some crowdsourcing locations may end up with a larger cost than the expectation. We would like to show that there exists some number λ such that for every crowdsourcing location, $Pr[c_i x_i \ge \lambda\theta^\star] \le \frac{1}{nm}$. Then, by the union bound, $Pr[\exists l_i, \sum c_i x_i \ge \lambda\theta^\star] \le \frac{|L|}{mn} \le \frac{1}{n}$. That is, we would get a λ approximation with a $\frac{1}{n}$ probability to exceed the $\lambda\theta^\star$. In the following, we will prove that $\lambda = O(\frac{\log n}{\log\log n})$.

Theorem 5. *The proposed algorithm has an $O(\frac{\log n}{\log\log n})$ approximation ratio.*


(a) San Franciso

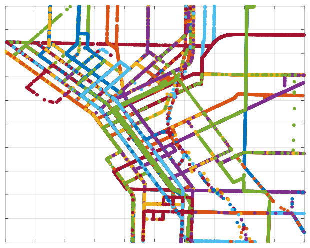

(b) Seattle

Fig. 6. The vehicles' movement history.

Proof. Without loss of generality, let us assume that all rounds of the randomized rounding algorithm are all disjoint events. Workers are selected with probabilities $\{x_1^\star, x_2^\star, ..., x_n^\star\}$ for any crowdsourcing location l_i, where $x_i^\star$ follows the independent and identically distributed random distribution, the probability that $\sum x_i^\star$ is normalized to 1. Let random variables $x_i \in [0,1]$. Then, $E[\sum c_i x_i] = \sum_1^n c_i E[x_i] =: \theta^\star$. Then, for any $\lambda > 0$ by using Chernoff bounds, we get the following result due to the random selection:

$$Pr[\sum c_i x_i \ge (1+\lambda)\theta^\star] \le \left(\frac{e^\lambda}{(1+\lambda)^{1+\lambda}}\right)^{\theta^\star} \tag{5}$$

It is equivalent to

$$Pr[\sum c_i x_i > \lambda\theta^\star] \le (\frac{e^{\lambda-1}}{\lambda^\lambda})^{-\lambda\theta^\star} \le (\lambda/e)^{-\lambda} \tag{6}$$

If we set $\lambda^\lambda \approx n$, that is, $\lambda = O(\frac{\log n}{\log\log n})$. Eq. 6 $\le \frac{1}{n}$. $\square$

VI. Performance Evaluation

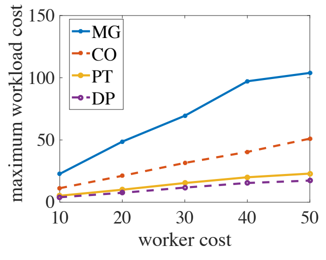
A. Traces

The EPFL [21] trace is the taxi trace collected from San Francisco, USA. It contains GPS coordinates of approximately 500 taxies over 30 days. Another trace that we use is the Seattle bus [22] trace. The traces were collected from the bus while on different routes in Seattle, USA for several weeks.

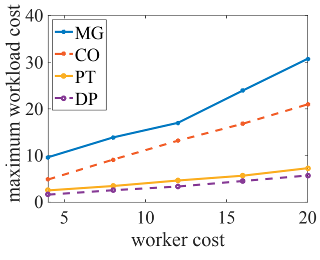
Some detailed experiment parameters are as follows: we choose the center city of these two cities, 10,000 (ft) $\times$ 10,000 (ft), as the experiment area. Then, we divide the experiment area into grids. The grid size is 200 (ft) $\times$ 200 (ft), which is the typical WiFi range under 2.4 GHz in 802.11 protocol for outdoor environment [23]. The experiment area and the vehicles' movement history are shown in Figs. 5 and 6. We consider that once a vehicle reaches a grid, it can successfully finish the crowdsourcing task. In the experiment, we choose the first 40 taxis in the EPFL trace, and we choose 236 buses in the Seattle bus trace. Since we do not have the cost information in these two traces, we generate five different costs, which refers from 5 different cost in Uber cars [24], i.e., Uber pool, UberX, UberXL, UberSelect, and UberBlack, in reality.

B. Experiment Setting

In the 1-D scenario, we randomly select a row in the area to conduct the experiments. The crowdsourcing locations are randomly selected among the grids [4, 20] and [2, 6],

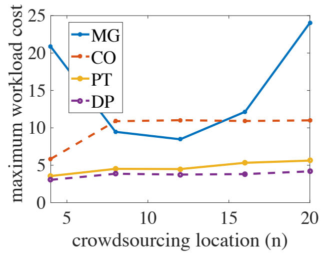


(a) San Francisco

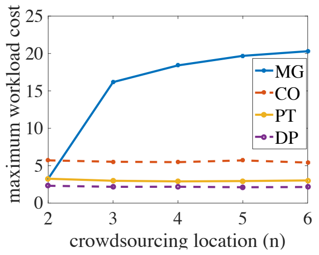


(b) Seattle

Fig. 7. Different cost ranges.



(a) San Francisco



(b) Seattle

Fig. 8. Different amounts of crowdsourcing locations.

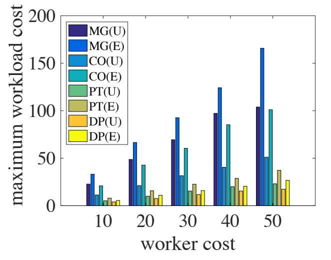
respectively in two datasets. For each vehicle, we use the uniform distribution and the exponential distribution to assign a cost for a vehicle. The enter position and the exit position are considered as the left-most grid and the right-most grid, respectively. In the 2-D scenario, we randomly select [5, 25] and [4, 20] crowdsourcing locations in the 2-D scenario. In addition, we use the uniform distribution to assign the vehicle cost. All the experiments are repeated 500 to 2000 times.

C. Algorithm Comparison

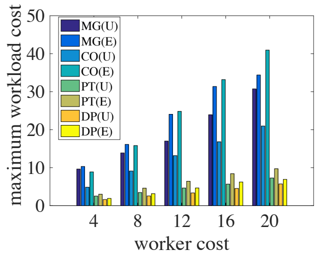
We propose four algorithms in the 1-D scenario. (1) *min-max greedy* (MG) algorithm selects the worker candidate sets which increase the max-cost of any crowdsourcing location at least, then among them, the worker who can increase the coverage most is selected until the network is covered. (2) *coverage-only* (CO) algorithm selects a set of workers from one side to another side without considering their cost. (3) *PTAS* (PT) algorithm divides the workers into to two sets according to their corresponding cost, then, uses worker with the lowest cost to cover the network. Then, we try to find the optimal set partition. (4) *dynamic programming* (DP) algorithm uses the dynamic programming technique to find the optimal solution. In the 2-D scenario, we extend the solution in the 1-D scenario and compare their performances with the *randomize rounding* (RD) algorithm, which uses the randomized rounding technique to select the workers.

D. Experimental results

1) 1-D scenario: Figs. 7, 9, and 8 show the result in the EPFL trace and in the Seattle trace in the 1-D scenario. In Fig. 7(a), the results show that along with the worker cost increase, the performance difference between the proposed four algorithms increases. In detail, the maximum workload



(a) San Francisco

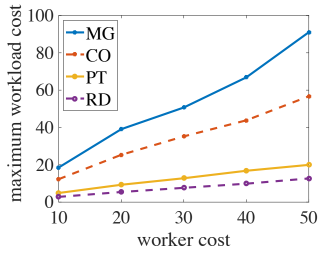


(b) Seattle

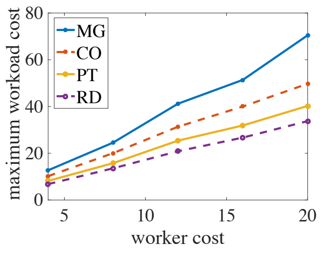
Fig. 9. Different cost distributions.

cost for any crowdsourcing location in the network decreases from the MG algorithm, followed by the CO algorithm and the PT algorithm. The DP algorithm achieves the lowest maximum workload cost, and the MG algorithm has the worst performance. The CO algorithm reduces 120% of the maximum cost. The PT algorithm further increases about 100% of the performance. The DP algorithm only further improves about 10 % of the performance with a higher computation complexity. This result demonstrates effectiveness of the proposed PT algorithm. For two different cost distribution cases, Fig. 9(a) shows the result, where algorithm(U) denotes the uniform cost distribution and algorithm(E) denotes the exponential distribution. It shows that exponential distribution leads to a worse performance. Figs. 7(b) and 9(b) show similar results in the Seattle trace. The difference is that there is a best point for the MG algorithm in EPFL trace, which shown in Fig. 8(a). The insight behind that is when the amount of crowdsourcing locations is small, an improper step might have a big influence on the future. When the amount of crowdsourcing locations is large, the improper selection accumulates. In the Seattle trace as shown in Fig. 8(b), the CO, PT, and DP performances remain nearly the same.

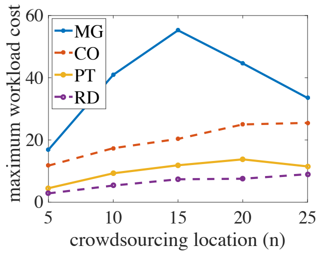
2) 2-D scenario: Fig. 10 shows the result in the EPFL and Seattle traces in the 2-D scenario. Figs. 10(a) and 10(b) show the similar performance order of the four algorithms in terms of different worker costs. That is, in Figs. 10(a) and 10(b), the maximum workload cost decreases following the order of the MG, CO, PT and RD algorithm. However, in Fig. 10(a), the CO and RD algorithms achieve 30% maximum workload cost than the MG algorithm in the EPFL trace. However, in the Seattle trace, the CO and RD algorithms achieve 50% maximum workload cost than that of the workload of the other two algorithms. The reason might be that the taxies visit the whole grid area more uniformly than the buses. Therefore, we can more easily to find a taxi with a smaller cost to cover the same area. However, different buses' routes are more different. In terms of the different amount of crowdsourcing locations, the proposed RD and PT algorithms also achieve the best and the second best performances. For the MG algorithm, the maximum workload cost gradually increases first and then decreases. The insight behind it is that when the crowdsourcing location is sparse, the improper selection of the MG algorithm increases. However, then the crowdsourcing location becomes dense, and the improper of the MG algorithm decreases, since it becomes easy to cover some uncovered areas. We also



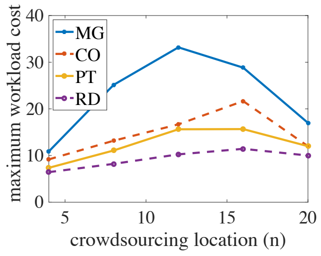
(a)



(b)

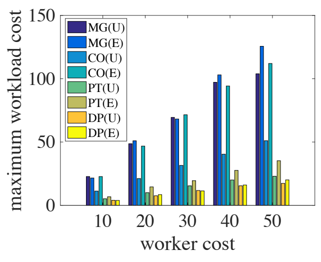


(c)

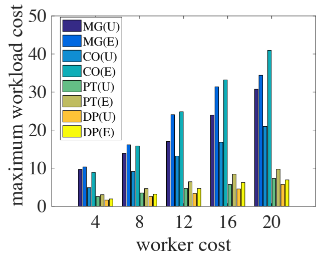


(d)

Fig. 10. Performance comparison in two datasets in the 2-D scenario.



(a) San Francisco



(b) Seattle

Fig. 11. Different cost distribution.

conduct experiments in terms of different cost distribution, Fig. 11 shows that the exponential cost distribution increases the maximum workload cost in the 2-D scenario.

VII. Conclusion

In this paper, we address coverage and workload-balancing in spatial crowdsourcing, which are ignored in existing approaches. We propose a series of worker recruit algorithms. We discuss the problem in the 1-D scenario first and propose a directional coverage algorithm with a performance bound of 2 times of maximum cost ratio. A PTAS extension is further proposed to improve the performance. At the end, we propose to use the dynamic programming approach to find the optimal solution in the 1-D scenario. Then, we address the problem in the general 2-D situation by using the randomized rounding algorithm. It achieves an $O(\frac{\log n}{\log \log n})$ approximation ratio in a high probability. Extensive experiments show that the proposed algorithms achieve a good performance.

References

[1] Y. Zhao and Q. Han, "Spatial crowdsourcing: current state and future directions," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 102–107, 2016.

[2] https://www.openstreetmap.org.

[3] https://www.waze.com.

[4] J. She, Y. Tong, and L. Chen, "Utility-aware social event-participant planning," in *Proceedings of the ACM SIGMOD ICMD*, 2015.

[5] Z. He and D. Zhang, "Cost-efficient traffic-aware data collection protocol in vanet," *Ad Hoc Networks*, 2016.

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

[7] J. She, Y. Tong, L. Chen, and C. C. Cao, "Conflict-aware event-participant arrangement," in *Proceedings of the IEEE ICDE*, 2015.

[8] W. Sun, Q. Li, and C.-K. Tham, "Wireless deployed and participatory sensing system for environmental monitoring," in *Proceedings of the IEEE SECON*, 2014.

[9] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, "Cartel: a distributed mobile sensor computing system," in *Proceedings of the ACM SenSys*, 2006.

[10] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the ACM SenSys*, 2008.

[11] F. Saremi, O. Fatemieh, H. Ahmadi, H. Wang, T. Abdelzaher, R. Ganti, H. Liu, S. Hu, S. Li, and L. Su, "Experiences with greengpsfuel-efficient navigation using participatory sensing," *IEEE Transactions on Mobile Computing*, vol. 15, no. 3, pp. 672–689, 2016.

[12] Y. Chon, N. D. Lane, Y. Kim, F. Zhao, and H. Cha, "Understanding the coverage and scalability of place-centric crowdsensing," in *Proceedings of the ACM Ubicomp*, 2013.

[13] X. Zhang, Z. Yang, Y. Liu, and S. Tang, "On reliable task assignment for spatial crowdsourcing," *IEEE Transactions on Emerging Topics in Computing*.

[14] B. Guo, Y. Liu, W. Wu, Z. Yu, and Q. Han, "Activecrowd: A framework for optimized multitask allocation in mobile crowdsensing systems," *IEEE Transactions on Human-Machine Systems*, 2016.

[15] H. Zheng, N. Wang, and J. Wu, "Minimizing deep sea data collection delay with autonomous underwater vehicles," *Journal of Parallel and Distributed Computing*, vol. 104, pp. 99–113, 2017.

[16] N. Wang and J. Wu, "Opportunistic wifi offloading in a vehicular environment: Waiting or downloading now?" in *Proceedings of the IEEE INFOCOM*, 2016.

[17] D. Zhao, H. Ma, and L. Liu, "Energy-efficient opportunistic coverage for people-centric urban sensing," *Wireless networks*, vol. 20, no. 6, pp. 1461–1476, 2014.

[18] D.-H. Shin, S. He, and J. Zhang, "Joint sensing task and subband allocation for large-scale spectrum profiling," in *Proceedings of the IEEE INFOCOM*, 2015.

[19] Y. Niu, L. Pan, M. J. Pérez-Jiménez, and M. R. Font, "A tissue p systems based uniform solution to tripartite matching problem," *Fundamenta Informaticae*, vol. 109, no. 2, pp. 179–188, 2011.

[20] P. Raghavan and C. D. Tompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[21] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "A parsimonious model of mobile partitioned networks with clustering," in *Proceedings of the IEEE COMSNETS*, 2009.

[22] J. G. Jetcheva, Y.-C. Hu, S. PalChaudhuri, A. K. Saha, and D. B. Johnson, "Design and evaluation of a metropolitan area multitier wireless ad hoc network architecture," in *Proceedings of the IEEE HotMobile*, 2003.

[23] http://www.wi-fi.org/.html.

[24] https://en.wikipedia.org/wiki/Uber_(company).