MC^2: A Framework for Weather Crowdsourcing

Hien To
Computer Science Dept.
University of Southern California
hto@usc.edu

Luan Tran
Computer Science Dept.
University of Southern California
luantran@usc.edu

Cyrus Shahabi
Computer Science Dept.
University of Southern California
shahabi@usc.edu

ABSTRACT

Spatial Crowdsourcing (SC) is a novel and transformative platform that engages individuals, groups and communities in the act of collecting, analyzing, and disseminating environmental, social and other spatio-temporal information. In this study, we introduce a framework for crowdsourcing weather information, in which workers that are within a certain distance from the task, e.g., workers within a distance of 5 km, are eligible to report weather. The objective of the framework is to minimize the number of assigned workers while maximizing the number of performed tasks. We investigate strategies for task assignment that balance two aspects of the framework, including communication cost (i.e., sending push notification to selected workers) and time to complete (i.e., delay of the tasks). Extensive experimental results on real-world datasets show that the proposed algorithms are efficient.

1. INTRODUCTION

With the ubiquity of smart phones and wireless network bandwidth improvements, every person with a mobile phone can now act as a multi-model sensor collecting and sharing various types of high-fidelity spatiotemporal data instantaneously (e.g., picture, video, audio, location, time, speed, direction, acceleration). Exploiting this phenomena, a new framework for efficient and scalable data collection has emerged, namely *spatial crowdsourcing* [27]. In our problem, the goal is to crowdsource a set of *spatial tasks* to a set of *workers*, which requires the workers to physically present in the task regions in order to perform the tasks. Note that each task is related to an area while each worker is associated with a location. To illustrate, consider an environmental sensing scenario, where a researcher (i.e., *requester*) is interested in collecting rainfall at various areas. First, the requester issues a query to a spatial crowdsourcing server (*SC-server*) asking for weather at different areas. Consequently, the SC-server crowdsources the query among the available workers in the vicinity of those areas. Once the workers report rainfall with their mobile phones, the results are sent back to the requester. This new paradigm for data collection can reduce costs significantly, and is particularly useful in the case of disaster response or censorship, when traditional means fail.

In [27], workers are required to physically travel to the task location (i.e., a point) in order to perform the task. However, in this problem each task is associated with a region so that any worker within the region is eligible to perform the task. There are two main distinctions. First, our problem one response from a worker can be used for multiple task requests as long as the tasks' regions cover the worker while in [27] each worker response is associated with only one requester. This is because the results for all weather crowdsourcing tasks at the same location are equivalent, i.e., two weather reports at the same location can be shared between two researchers. Thus, task assignment is no longer a matching problem [27] but a selection problem, i.e., selecting the workers to answer crowdsourcing tasks. Second, in [27] the communication cost of sending task requests to workers (i.e, indicated by the number of selected workers) was not considered though achieving maximum assigned tasks would require a large number of workers. However, minimizing the communication cost is critical for any mobile application, particularly at large scale. In the new problem setting, we show that optimum task assignment can be achieved while also minimizing the the number of selected workers.

We introduce a task assignment problem, namely, maximum coverage minimum cost (MC^2), in which the optimization goal is to select the minimum number of workers such that the total number of covered tasks is maximized. We prove that MC^2 is a NP-hard problem by reducing to geometric hitting set. The solution to the MC^2 could be straightforward if the SC-server had a global knowledge of both the spatial tasks and the workers. Existing algorithms such as greedy can be used, which provides best-possible polynomial time approximation. However, the server is continuously receiving spatial tasks from requesters and updated locations of the workers. Therefore, the server can only apply the algorithm at every time instance (i.e., local optimization) with no knowledge of the future.

We proposed four alternative solutions to the MC^2

problem. The first approach, namely *High Task Coverage Priority (HTCP)*, follows the simple local optimization strategy by maximizing the task assignment at every time instance. *HTCP* chooses the workers that covers the greatest number of unassigned tasks at each stage. Our second approach, called *Low Worker Coverage Priority (LWCP)*, improves the overall task assignment by assigning higher priority to spatial tasks whose regions cover less workers. The intuition is that spatial tasks are more likely to be performed in future if they are located in worker-dense areas (i.e., areas with high population of workers). Our third approach exploits a particular property of the problem space. The intuition is that the number of assigned workers is minimized if each worker response can be used for as many tasks as possible. We referred the number of tasks can be solved by a worker response as *fan-out*. Thus, the third approach, namely *Large Worker Fan-out Priority (LWFP)* selects a worker iff its fan-out is larger than a threshold or the task will be expired at next time instance. The fourth approach, namely *Close-To-Deadline Priority (CTDP)*, captures the task deadline by giving higher priority to tasks whose deadlines are close to the deadline. At each stage, *CTDP* chooses a worker with smallest average time to deadline of its unassigned tasks.

Thus our specific contributions are:

(i). We propose a framework for crowdsourcing weather sensing tasks, which are distinct from generic spatial crowdsourcing [27] in several aspects, applications, design goals, and performance measurements. The framework has applications in environmental sensing and climate studies.

(ii). We formally define maximum coverage minimum cost (MC^2) problem in weather crowdsourcing and show the hardness of the problem by reducing it to geometric hitting set.

(iii). We propose alternative solutions to MC^2 by exploring the spatial and temporal aspects of the problem. To improve quality of task result, we extends MC^2 to redundant task assignment where a task request requires multiple responses from workers.

(iv). Our analysis and extensive experiments show that we can always obtain the maximum number of assigned tasks while minimize the number of selected workers. In comparison with *HTCP* (baseline), our heuristics (i.e., *LWCP*, *LWFP* and *CTDP*) shows its superiority in minimizing the number of selected workers.

The remainder of this paper is organized as follows. In Section 2, we present the proposed framework and its applications. In Section 3, we discuss a set of preliminaries in the context of weather crowdsourcing. Thereafter, in Section 4, we formally define the MC^2 problem, then explain our assignment solutions. Section 5 reports our experimental results. In Section 6, we review the related work. Finally, Section 7 concludes and discusses the future directions of this study.

2. CROWDSOURCING FRAMEWORK

Section 2.1 presents the system model and the workflow for weather crowdsourcing framework. Section 2.2 discusses design challenges and associated performance metrics.

2.1 System Model

According to [27], there are two categories of SC, based on how workers are matched to tasks. In *Worker Selected Tasks (WST)* mode, the SC-server publishes the spatial tasks online, and workers can autonomously choose any tasks in their vicinity without the need to coordinate with the server. In *Server Assigned Tasks (WST)* mode, online workers send their location to the server, and the server assigns tasks to nearby workers.

WST is a simpler protocol; however, the assignment is often sub-optimal as workers do not have a global system view. Workers typically choose some nearby tasks to them, which may cause multiple workers to perform the same task *unnecessarily*, while many other tasks remain unassigned. We consider the problem of *task assignment* in the SAT mode. The SAT mode incurs the overhead of running complex assignment algorithms at the SC-server, but the best-suited workers are selected for a set of tasks. This requires the SC-server to know the workers' locations, which poses a privacy threat. The privacy issue is beyond the scope of this paper[1].

Figure 1 shows the proposed system architecture. In Step 0, workers send their locations to the SC-server. When the SC-server receives a set of tasks t_i (Step 1), it selects a minimum set of workers that covers maximum number of tasks. Next, the SC-server initiates a *push notification* process (Step 3) to disseminate t_i to the corresponding workers within the task regions. As mentioned, each task is associated with a region such that all workers within the task region that are eligible to perform task[2]. In other words, one worker's response can be used for multiple tasks whose regions cover the worker location.

In our framework, both tasks and workers can come and go without our knowledge. That is, the SC-server is continuously receiving tasks from requesters and updated locations from the workers. Therefore, the server

[1]A solution to protecting workers' locations in spatial crowdsourcing was proposed in [41].

[2]The word *cover* is used in both ways, a task region covers a worker and a worker covers a task

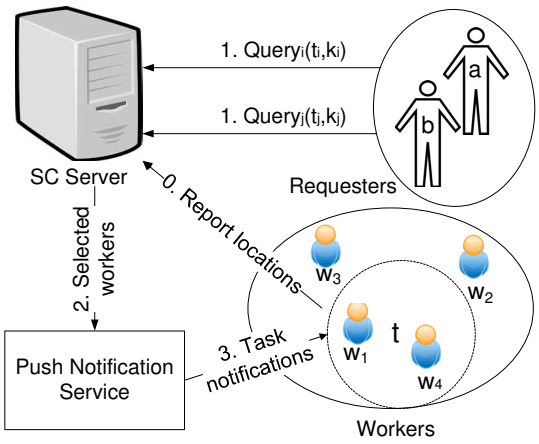


Figure 1: The spatial crowdsourcing framework.

can only maximize the task assignment at every time instance (i.e., local optimization) with no knowledge of the future.

We assume that all tasks are micro and take the same amount of time to complete. To ensure the tasks will actually be fulfilled, these workers are required to perform their assigned tasks. In addition, as the worker responses may be incorrect, every task is required to be performed by a particular number of workers. The higher the number of responses, the more validity the result is. One of the well-known mechanisms to make a single decision based on the results provided by a group of workers is majority voting. This method accepts the result supported by the majority of workers, which is likely to be trusted.

2.2 Design Goals and Performance Metrics

Our specific objective is to maximize the number of assigned tasks while minimizing the number of selected workers. Therefore, we focus on the following performance metrics:

- **Assignment Success Rate (ASR).** Maximizing the number of assigned task is our priority. However, due to multiple time instances, the SC-server may not achieve the maximum task assignment rate (i.e., a task region is only covered by workers in current time instance but not in the following time instances as the worker set may be different per time instance). *ASR* measures the ratio of the number of tasks accepted by workers to the total number of task requests. Note that *ASR* does not capture worker reliability, tasks may still fail to complete after being accepted. Reliability of task assignment can be improved by having a task performed my multiple workers.

- **Communication Overhead.** That one worker can response to multiple tasks increases the complexity of the assignment algorithms, which poses scalability problem. A significant metric to measure overhead is the average number of notified workers (**ANW**). This number represents the *communication overhead* required to send tasks to the workers[3].

- **Real-time Assignment.** A desired objective of any crowdsourcing application is to enable real-time task assignment. This is particular useful in weather crowdsourcing application, in which weather condition, e.g., rain, drought, flood, could be reported in real-time for various applications, e.g., disaster response, travel decision making. A reasonable measurement is the average time delay (**ATD**), which indicates how fast the tasks are assigned. Minimize both communication overhead and ATD at the same time is challenging as they are contradict. This is because task assignment is more effective in terms of communication cost if the tasks are delay till their deadlines so that one one worker response can be used for many task requests.

2.3 Applications

Our framework has various applications, e.g., environmental sensing, disaster response and travel decision making. In the following, we present a specific application of the framework for real-time rainfall observation. Precipitation information is a key variable in water resources management, hazard preparedness, and climate studies. Satellite remote sensing technologies [21] have been utilized in precipitation estimation and monitoring since 1960s. As an indirect measurement method, satellite precipitation estimation is challenging and often associated with uncertainties. In collaboration with scientists at the Center for Hydrometeorology and Remote Sensing[4] at the University of California, Irvine, we utilize spatial crowdsourcing technology to enable human workers to report precipitation condition, particularly rain levels (e.g., heavy/medium/light/none) observation to improve real-time global satellite precipitation estimation[5]. How to use human-reported precipitation information to improve [21] is beyond the scope of this paper. We collect rainfall information through two ways. First, workers report rainfall level with spatio-temporal information to the server when they see rain occurring around their locations (i.e., volunteered geographic information). The other way is via spatial crowdsourcing, in which researchers can post a set of

[3]In practice, a task can be sent to assigned workers in real-time using push notification. This may may be costly at large scale, e.g., Parse.com, a cloud-based mobile app platform for push notifications, charges $0.05 per thousand push notifications.

[4]http://chrs.web.uci.edu/

[5]http://hydis.eng.uci.edu/gwadi/

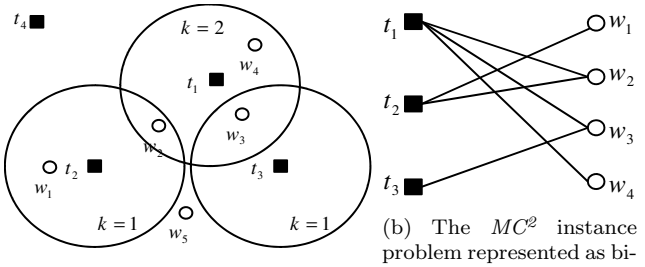
spatial tasks, such as collecting rainfall levels at some particular places during a time period. Then, the SC-server crowdsources the tasks to a set of workers that are physically present at the tasks' regions. Based on our proposed framework, we built iRain, a mobile system for crowdsourcing precipitation information. iRain consists of a database server, an application server and mobile apps for Android[6] and iOS phones[7]. Besides rainfall, our framework can be easily tailored for other weather conditions, such as e.g., freezing rain, snow, drought, drizzle, air pollution.

3. PRELIMINARIES

In this section, we define a set of terminology in the context of SAT mode. First, we formally define a *spatial task*.

DEFINITION 1 (SPATIAL TASK). *A spatial task t of form $<l, r, s, \delta>$ is a query to be answered within a circle (l, r)[8], where task location l is the center point in the 2D space and r is radius. The query is asked at time s and will be expired at time $s + \delta$.*

Figure 2a shows the spatial regions of three tasks, t_1, t_2, t_3. Each task defines a circle where enclosing workers are eligible to perform the task. On the other hand, a worker only report to a task if he is within a distance from the task location. For example, w_2, w_3, w_4 are eligible to perform t_1. An example of a spatial task is to report weather condition (e.g., light/medium/heavy rain, or snow). In the following we formally define a worker.



(a) An example of W_i and T_i

(b) The MC^2 instance problem represented as bipartite graph $G_i(V, E)$

Figure 2: An example of workers and tasks snapshot in time instance s_i

DEFINITION 2 (WORKER). *A worker, denoted by 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 online when he is ready to accept tasks. Note that when the worker reports to a task request, the result can be used for all other tasks whose task regions enclose the worker and deadlines have not been passed. Once a worker goes online, he sends his updated location to the SC-server (Figure 1).

Since the task assignment is not a one-time process, the SC-server continuously receives SC-queries from requesters and updated locations from workers at every time instance. Therefore, we define the notion of coverage instance set.

DEFINITION 3 (COVERAGE INSTANCE SET). *Let $W_i = \{w_1, w_2, ...\}$ be the set of online workers at time s_i. Also, let $T_i = \{t_1, t_2, ...\}$ be the set of available tasks at time s_i. The coverage instance set, denoted by I_i is the set of tuples of form $< w_j, ts_{w_j} >$, where the worker w_j is covered by every task region in the task set ts_{w_j}. Also, $|I_i|$ denotes the number of tasks, which are covered at time instance s_i.*

The coverage instance set must conform to the constraints of the tasks. That is, for every tuple $<w_j, ts_{w_j}> \in I_i$, the spatial constraint ensures the worker w_j covers all tasks in the task set ts_{w_j} while the temporal constraint means the tasks are not expired.

With the assumption that each task only requires one response, we now define the *maximum coverage minimum cost* problem.

DEFINITION 4 (MC^2). *Given a time interval $\phi = \{s_1, s_2, ..., s_n\}$, the Maximum Coverage Minimum Cost (MC^2) problem is the process of selecting the minimum number of workers during the time interval ϕ such that the total number of assigned tasks $\sum_{i=1}^{n} |I_i|$ is maximized.*

In the ideal case, a minimum number of workers cover all tasks. However, this might not be practical due to the constraints of the tasks and workers, i.e., some tasks may not be covered by any worker or some tasks may be covered by many worker.

4. ASSIGNMENT PROTOCOL

The main challenges of spatial crowdsourcing are due to the large-scale, ad-hoc and dynamic nature of the workers and tasks. First, to continuously match thousands of spatio-temporal tasks, with millions of workers, the SC-server must be able to run efficient worker selection strategies that can scale. Second, the worker selection must be performed frequently and in real-time as new tasks and workers become available or as tasks are completed (or expired) and workers leave the system.

In order to optimally solve the MC^2 problem, the SC-server should have a global knowledge of all the spatial tasks and the workers. However, the server does not have such knowledge as at every time instance, it receives a set of new tasks from the requesters and a set

[6]https://play.google.com/store/apps/details?id=com.irain
[7]https://itunes.apple.com/us/app/irain-app/id882089312
[8]Task region can be extended to any shape, e.g., rectangle

of new locations from the workers. Therefore, the server only has a local view of the available tasks and workers at any instance of time. This means that a global optimal selection is not feasible. Instead, the server tries to optimize the worker selection locally at every instance of time. In the following, we propose a solution to one MC^2 instance problem.

4.1 MC^2 Instance Problem

Given a set of online workers $W_i = \{w_1, w_2, ...\}$, and a set of available tasks $Ti = \{t_1, t_2, ...\}$ at one time instance s_i, the goal is to select minimum number of workers that maximize the number of covered tasks, which is equivalent to maximizing $|I_i|$. We refer to this as maximum coverage minimum cost (MC^2) instance problem. The idea of solving this problem is to utilize the spatial constraints of tasks to ensure that workers are properly selected. With the following theorem, we first solve the MC^2 instance problem by reducing it to the *geometric hitting set* problem.

THEOREM 1. *The maximum coverage minimum cost instance problem is reducible to the geometric hitting set problem, which is NP-hard.*

PROOF. In the geometric hitting set problem [37], given a set of geometric objects and a set of points, the goal is to compute the smallest subset of points that hit all geometric objects. The problem is formalized as follows. Let X be the finite set of $|X|$ points and Y be the finite set of $|Y|$ regions in R^2, e.g., circles, rectangles $(y_1, y_2, ..., y_m)$. The goal is to find the smallest sized hitting set or set cover. Note that the geometric hitting set and geometric set cover was known to be dual to each other [5].

The MC^2 instance problem can be reduced to the geometric hitting set problem by simple reduction as follows. First, removing the tasks that do not cover any worker from the available task set T_i, the new task set is T_i'. Next, let consider the online workers at time instance s_i, W_i as the set of points X and the set of the task regions of the new task set T_i' as the set of regions Y in the geometric hitting set problem, the goal of MC^2 instance problem becomes to find the smallest subset of points in Y that hits all task regions. By removing the tasks that do not cover any worker, the reduction ensures that there is a subset of points hitting all task regions.

□

Figure 2 better clarifies this reduction. The task region of t_4 does not cover any worker, thus it is removed from current time instance. Note that t_4 may still be available in the next time instance if its deadline is not passed. The workers become a set of points $X = \{w_1, w_2, w_3, w_4, w_5\}$ and the remained task regions Y in geometric hitting set problem, $Y = \{r_{t_1}, r_{t_2}, r_{t_3}\}$. The hitting sets includes, $\{r_{t_1} \odot^T (w_2, w_3, w_4), r_{t_2} \odot^T (w_1, w_2), r_{t_3} \odot^T (w_3)\}$, where the $\odot^T$ represents hitting operator, e.g., r_{t_1} hits w_2, w_3 and w_4. The hitting sets are equivalent to the covering sets $\{w_1 \odot (r_{t_2}), w_2 \odot (r_{t_1}, r_{t_2}), w_3 \odot (r_{t_1}, r_{t_3}), w_4 \odot (r_{t_1})\}$, where $\odot$ represents covering operator, e.g., w_1 covers r_{t_2}. This is geometric set cover problem, which select a minimum set of workers that cover all tasks.

The geometric set cover and hitting set problems have been extensively studied in literature [19, 12, 36, 37, 15, 7, 8, 4, 5], also the non-geometric variants [11, 22, 9, 24, 30]. The problem is known to be strongly NP-hard even for simple geometric objects like unit disks in the plane. Therefore, unless P=NP, it is not possible to achieve optimal solution in polynomial time for such problem. Unsurprisingly, most studies focus on improving lower-bound of polynomial-time approximation algorithms for these problems. Typically, [11] presents a greedy algorithm, which achieves a lower-bound on approximation ratio $ln(n)$, where n is the number of sets. Later study [31] shows that greedy algorithm is the best-possible polynomial time approximation algorithm for set cover/hitting set. Other studies in geometric settings (e.g., [12]) improve approximation factor. More related works are discussed in Section 6. The hardness of the problem is not the focus of this study. Instead, we pick the greedy algorithm[11] as the baseline for solving MC^2 instance problem and focus on improving the algorithm by exploiting spatial and temporal properties of the tasks at multiple time instances.

4.2 Heuristics to MC^2

As mentioned earlier, a global solution to MSA is not feasible. Instead, the SC-server tries to optimize worker selection locally at every instance of time. Thus, our strategies solves the MC^2 problem by solving the MC^2 instance problem for every instance of time. In the following, we present a local optimum strategy to solve the MC^2 problem, namely *HTCP*, and three heuristics that improves *HTCP*.

4.2.1 High Task Coverage Priority (HTCP)

The basic approach solves the MC^2 problem by using greedy heuristic [11] for every time instance. At each stage, *HTCP* selects the worker that covers the maximum number of unassigned tasks. For instance, in Figure 2b, either w_2 or w_3 would be selected at the first stage. Algorithm 1 presents details of *HTCP*. Particularly, to construct input for MC^2 instance in Line 7, pre-process steps are performed in in Lines 4-6. Line **??** removes expired tasks from uncovered tasks carried from previous time instance (i.e., tasks with $s + \delta > i$). Line 5 adds uncovered non-expired tasks to current task set. In Line 7, each worker w is associated with a set of

tasks whose regions cover the worker, denoted by $\{w\}$. In Line 10, a worker that covers maximum number of unassigned tasks is selected. In Line **??**, the result per time instance includes the selected workers and the corresponding covered tasks. These outputs will be used to evaluate the performance of the algorithms in Section 5. In Line 13, the uncovered tasks are carried to the next time instance. The algorithm terminates when all tasks are covered in Line 9.

Algorithm 1 BASIC ALGORITHM

1: Input: worker set W_i and task set T_i per instance, task radius r, task period δ
2: Output: the selected workers per time instance S_i
3: For each time instance i:
4: Remove expired tasks $U'_{i-1} \leftarrow U_{i-1}$
5: Update task set $T_i \leftarrow T_i \cup U_{i-1}$
6: Remove tasks that do not cover any worker $T'_i \leftarrow T_i$
7: Construct worker set W_i (each contains a task set $\{w\}$)
8: Initialize $S_i \leftarrow \{\}$, covered tasks $Q \leftarrow \cup_w^{W_i}\{w\}$
9: While $Q.size > 0$
10: Select w in W_i that maximize $|\{w\} \cap Q|$
11: $Q \leftarrow Q - \{w\}$; $S_i \leftarrow \{w\}$
12: Save selected workers S_i and covered tasks $C_i \leftarrow \cup_w^{S_i}\{w\}$
13: Keep uncovered tasks $Q_i \leftarrow T'_i - C_i$

4.2.2 Large Worker Fan-out Priority (LWFP)

This heuristic bases on an idea that the total number of assigned workers is minimized if each worker performs as many tasks as possible. Similar to *HTCP*, at each stage *LWFP* selects the worker that covers the maximum number of unassigned tasks. However, *LWFP* improves *HTCP* by selecting only workers that either cover at least m tasks or cover will be expired in the next time instance. With this strategy, many tasks may accumulate through a few time instances before they are covered. Thus, the average time delay (ATD) would be higher than previous approaches, *HTCP* and *LWCP*. The trade-off between ANW and ATD is determined by the controlling parameter m. The higher m, the smaller ANW but the larger ATD. With *LWFP*, only Line 10 in Algorithm 1 needs to be updated. The updated algorithm only considers the workers that cover at least m tasks or will be expired in the next time instance.

4.2.3 Close-To-Deadline Priority (CTDP)

Our observation is that tasks that are approaching their deadlines are more important than the others. Thus, the workers that cover important tasks are also important. The importance of a worker can be computed as the average time to deadline (i.e., deadline minus current time) of the unassigned tasks covered by the worker, referred to as *weight*. Consequently, instead of selecting the worker that covers the maximum number of unassigned tasks, *CTDP* improves *HTCP* by choosing the worker of highest *weight* at each stage. Unlike *LWFP*, *CTDP* improves Algorithm 1 by replacing Line 10 to select $w \in W_i$ whose weight is the highest.

4.2.4 Low Worker Coverage Priority (LWCP)

The second heuristic is based on an observation that a task is more likely to be performed in the future if it is located in an area with higher population of workers. Thus, similar to *HTCP*, *LWCP* also chooses the worker that covers the maximum number of uncovered tasks at each stage (i.e., there can be multiple workers satisfy the condition), but *LWCP* improves *HTCP* by selecting the worker whose associated tasks cover the smallest number of workers. In Figure 2b, the associated tasks of w_2 includes t_1 and t_2, which covers all workers. The associated tasks of w_3 includes t_1 and t_3, which covers w_2, w_3 and w_4. Thus, w_3 is selected at the first stage.

4.3 Spatial Improvement

The problem with the presented strategies is that they only tries to optimize locally at every instance of time. Even though we are clairvoyant on neither the tasks from the requesters nor the future locations from the workers, we can explode the spatial characteristics of the environment during the assignment, one of which is the distribution of the workers. The observation is that a task is more likely to be performed in the future if it is located in an area with higher population of workers. Therefore, the idea is to assign higher priority to tasks that are located in worker-sparse areas, so that those tasks can be assigned before the ones in worker-dense areas. Consequently, there are more tasks to be assigned.

We used *location entropy*, which was first introduced in [13] to measure the diversity of unique visitors of a location. Location entropy takes into account the total number of workers in the task's location as well as the relative proportion of their future visits to that location. A location has a high entropy if many workers visit that location with equal proportions[9]. In contrast, a location will have a low entropy if there are only a few workers visit. Thus, our heuristic is to give higher priority to tasks that are located in areas with smaller location entropy, because those tasks have lower chance of being completed by other workers.

We now formally define the location entropy. For a given location l, let O_l be the set of visits to location l. Thus, $|O_l|$ gives the total number of visits to l. Also, let W_l be the set of distinct workers that visited l, and O_{wl} be the set of visits that worker w has made to the location l. The probability that a random draw from O_l

[9]The concept of visiting the same location is not of primary concern in this paper but for the sake of completeness we assume some sort of a discretization of space (e.g., a grid) and thus a location is represented by a grid cell, which can be visited by several workers.

belongs to O_{wl} is $P_l(w) = \frac{|O_{wl}|}{|O_l|}$, which is the fraction of total visits to l that belongs to worker w. The location entropy for l is computed as follows:

$$Entropy(l) = -\sum_{w \in W_l} P_l(w) \times logP_l(w) \qquad (1)$$

By computing the entropy of every location, we can associate to every task t_i of form $<l_i, q_i, s_i, \delta_i>$ a certain cost, which is the entropy of its location l_i. Accordingly, tasks with lower costs have higher priority, since they have a smaller chance of being completed. Thus, our goal in this approach is to assign the maximum number of tasks during every instance of time while the total cost associated to the assigned tasks is the lowest. We refer to this problem as the minimum-cost maximum task assignment instance problem. With the following theorem, we can solve the minimum-cost MTA instance problem by reducing it to the minimum-cost maximum flow problem [29]. A minimum cost maximum flow of a network $G = (V, E)$ is a maximum flow with the smallest possible cost.

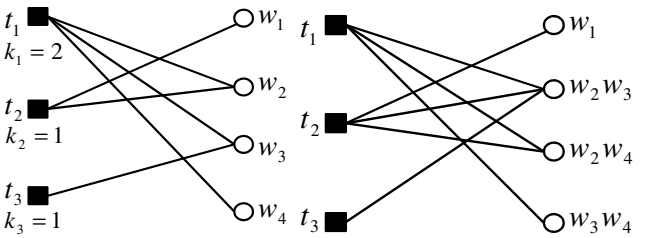
4.4 Redundant Assignment

In this section, we extend the MC^2 instance problem to redundant task assignment. Within one time instance, SC-server receives a set of tasks of form $<(t_1, k_1), (t_2, k_2), ...>$ from requesters, in which a spatial task t_i needs to be crowdsourced to a number of workers k_i, $k_i \geq 1$. If the task t_i is assigned to less than k_{t_i} workers, it is not counted. Having tasks performed redundantly by multiple workers increases the confidence on the responded results [28].

We will reduce redundant assignment problem to the single assignment by simple reduction. For every task whose k is larger than 1 (i.e., t_1 with $k_1 = 2$), we replace its hitting set, i.e., w_2, w_3, w_4 by all k-combinations of the set, i.e., $(w_2, w_3), (w_2, w_4), (w_3, w_4)$ (Figure 3). Each combination is a *logical* worker. Subsequently, we have a set of logical worker, $\{(w_2, w_3), (w_2, w_4), (w_3, w_4), (w_1), (w_2), (w_3)\}$. By removing the logical workers that are dominant by other logical workers, we have a new set $\{(w_2, w_3), (w_2, w_4), (w_3, w_4), (w_1)\}$ as shown in Figure 3b. We will prove that finding minimum worker cover set in the reduced space (Figure 3b) is equivalent to solving MC^2 instance problem (Figure 3a), in which the minimum worker set to MC^2 (i.e., w_2 and w_3) is the union of the minimum set of logical workers (i.e., (w_2, w_3)). [proof]

The number of k-combinations per task may be large $\binom{k}{n}$, where n is the number of workers that cover the task. Algorithm 2 presents the reduction that prunes the *dominated* logical workers when possible. A logical worker is dominated if its k-combination is covered by k-combination of another logical worker.

In Line 2, the tasks are sorted in descending order of the number of required responses is to enable pruning in Line 6. Sorting ensures that a logical worker is only dominated by prior logical workers but not in the other way. Similarly, the tasks are also sorted by the number of workers cover it so that pruning in Line 5 can be applied.



(a) $k \geq 1$ (b) $k = 1$

Figure 3: An example of the reduction of MC^2 instance problem with $k > 1$ to the case when $k = 1$.

Algorithm 2 REDUCTION ALGORITHM

1: Input: worker set W_i, task set T_i and corresponding required number of responses K_i
2: Sort T_i by K_i in *descending* order. For the tasks of the same k, they are sorted by the number of workers covering it.
3: For each task, populate its k-combinations (i.e., virtual workers). Then, connect the task to the virtual workers.
4: There are cases when k-combinations of a task are needn't to be populated. Instead, the task is connected to populated ones which contain a worker that covers the task. If one of the following condition satisfies.
5: Condition 1. If the worker set of a task of k responses is covered by the worker set of another traversed task of the same k.
6: Condition 2. If the worker set of a task is covered by an existing logical worker.
7: The minimum worker cover set is the union of the worker sets in the minimum set of logical workers.

5. PERFORMANCE EVALUATION

We conducted several experiments on both real-world and synthetic data to evaluate the performance of our proposed approaches. Below, we first discuss our experimental methodology. Next, we present our experimental results.

5.1 Experimental Methodology

We evaluated the scalability of our proposed approaches by varying radius R of the spatial region of the tasks. By varying the radius, we also vary the average number of workers that is eligible to perform a given spatial task, namely workers per task (W/T), and the average number of spatial tasks which can be performed by a given worker, denoted by tasks per worker (T/W). To evaluate the experimental results, we used two performance measures: the average number of assigned tasks per time instance and the average number of assigned workers per time instance. We conducted our experiments on two synthetic (SYN) and two real-world (REAL) data

	Gowalla	Yelp
No. of tasks	20,000	11,537
No. of workers	10,273	43,873
Avg. No. of W/T	4	44

Table 1: Statistics of the real data. This data is not from crowdsourcing application but rather the transformed data sets of the geospatial data sets in the spatial crowdsourcing context.

sets.

With our synthetic experiments we used two distributions: uniform (SYN-UNIFORM) and skewed (SYN-SKEWED). With the uniform distribution (SYN-UNIFORM), both workers and tasks are uniformly distributed. Whereas, in order to generate the SYN-SKEWED data set, the workers were formed into four Gaussian clusters (with $\sigma = 0.05$ and randomly chosen centers) while the tasks were uniformly distributed. As the results, the average number of W/T varies with a small standard deviation, whereas in our experiments on SYN-SKEWED, the average number of W/T varies with a large standard deviation.

Our real data sets include Gowalla[10] and Yelp[11]. A summary of these data sets is given in Table 1. Gowalla is a location-based social network, where users are able to check in to different spots in their vicinity. The check-ins include the location and the time that the users entered the spots. For our experiments, we used the check-in data over a period of 20 days in 2010, including 10,273 spots (e.g., restaurants), covering the state of California. Moreover, we assumed that Gowalla users are the workers of our spatial crowdsourcing system. We picked the granularity of a time instance as one day. Consequently, we assumed all the users who checked in during a day as our available workers for that day. Moreover, at each time instance, 1000 spatial tasks were randomly generated for the given spots in the area.

The Yelp data set was captured in the greater Phoenix, Arizona, including locations of 11,537 businesses (e.g., restaurants), 43,873 users and 229,907 reviews. For our experiments, we assumed that the businesses are the spatial tasks, Yelp users are the workers. To generate the time instances, we divided the time interval θ (i.e., review time interval) into 20 equal periods. Other parameters are shown in Table 1.

Finally, in all of our experiments, we fixed the time interval to 20 days and set the duration of every spatial task to 5 days (i.e., $\phi = 20, \delta = 5$). With the SYN experiments, we fixed the number of new workers and the number of new tasks at each instance time to 500 and 1000, respectively. Finally, experiments were run on an Intel(R) Core(TM)i7-2600 CPU @ 3.40 GHz with 8 GB of RAM.

5.2 Experimental Results

[10]snap.stanford.edu/data/loc-gowalla.html

[11]yelp.com/dataset_challenge

We evaluate performance of our approaches in terms of 1) the average number of assigned workers per time instance (i.e., communication cost) - the smaller the better 2) the average number of assigned tasks per time instance - the higher the better, 3) the average task per worker - the higher the better and 4) the average time to deadline of the assigned tasks - the smaller the better.

5.2.1 *Fixing worker set*

In the first set of experiments, we evaluated the performance of our heuristics by fixing the worker set for all time instances. Figure 4 shows that as the task region enlarges (i.e., R increases), the number of selected workers decrease greatly while the number of assigned tasks increase slowly. This results in the average task per worker increases. Particularly, *CTDP* is shown to be the best in terms of minimizing the communication cost. However, Figures 4d and 4h show that *CTDP* requires the highest time to deadline. On the other hand, *LWFP* efficiently balances the two objectives, communication cost and time to deadline.

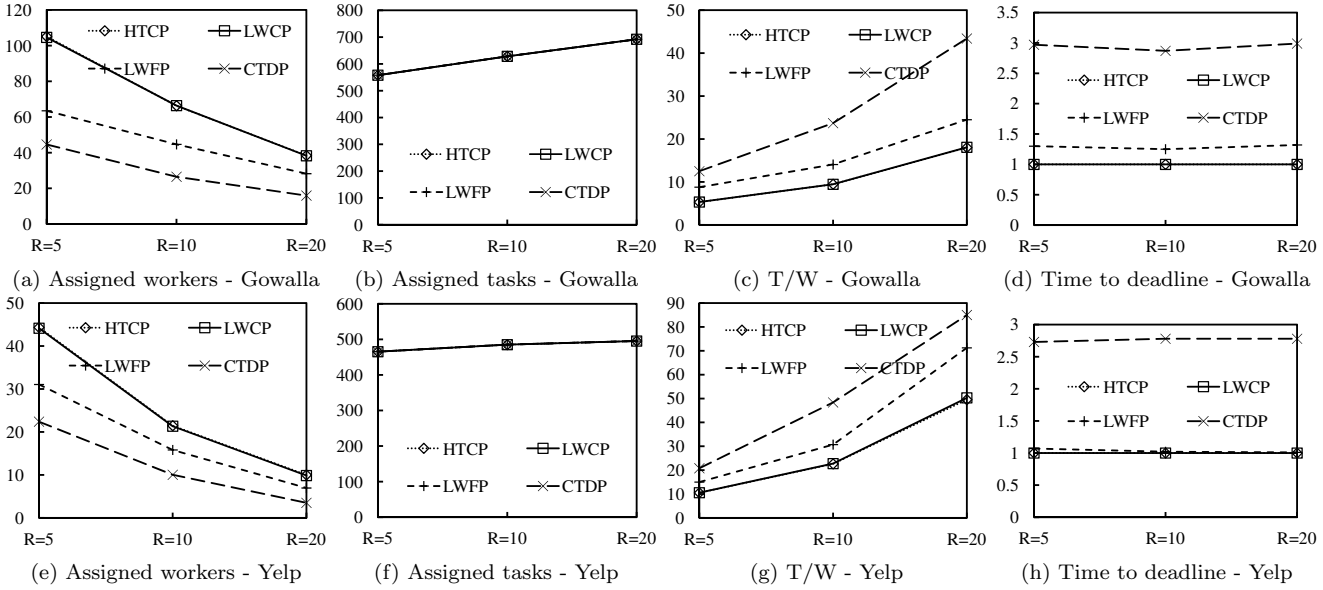
5.2.2 *Varying worker set*

In the second set of experiments, we evaluated the performance of our heuristics by varying the worker set for all time instances. Figure 5 shows similar trends as in Figure 4, excepts the number of assigned tasks is reduced in *CTDP*. This is because *CTDP* prefers tasks to be assigned when approaching their deadlines.

6. RELATED WORK

Spatial Crowdsourcing: While crowdsourcing has largely been used by both research communities (e.g., image processing [10], [40] and [43], databases [20], [33], [39] and [16]) and industry (e.g., oDesk [2], MTurk [1] and CrowdFlower [3]), spatial crowdsourcing only recently received some attention [41],[28], [14], [28] and [6]. In [6], a crowdsourcing platform is proposed, which utilizes location as a parameter to distribute tasks among workers. In [27], a spatial crowdsourcing framework whose goal is to maximize the number of assigned tasks is proposed. Since the workers cannot always be trusted, another work [28] aims to tackle the issue of trust by having tasks performed redundantly by multiple workers. In [14], the problem of complex spatial tasks (i.e., each task comprises of a set of spatial sub-tasks) is introduced, in which the assignment of the complex task requires performing all of its sub-tasks. Recently in [41], the authors introduced the problem of protecting worker location privacy in spatial crowdsourcing. This study proposes a framework that achieves differentially-private protection guarantees.

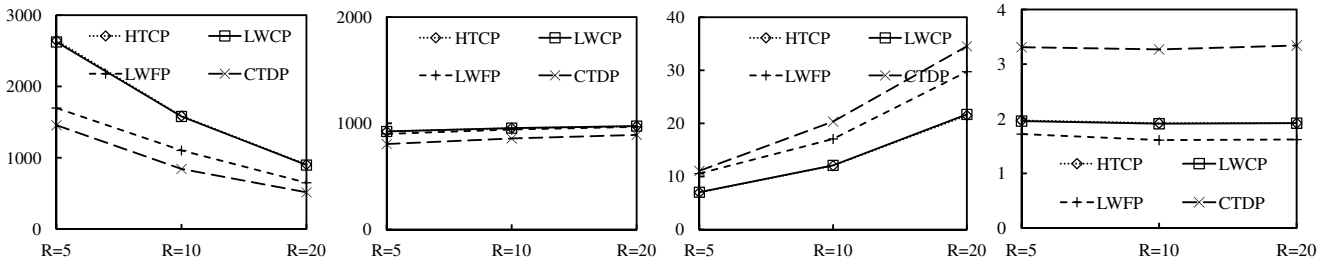
Participatory sensing: The well-known concept of participatory sensing could be deemed as one class of



(a) Assigned workers - Gowalla (b) Assigned tasks - Gowalla (c) T/W - Gowalla (d) Time to deadline - Gowalla

(e) Assigned workers - Yelp (f) Assigned tasks - Yelp (g) T/W - Yelp (h) Time to deadline - Yelp

Figure 4: Performance comparison of the proposed heuristics when fixing the worker set.



(a) Assigned workers - Gowalla (b) Assigned tasks - Gowalla (c) T/W - Gowalla (d) Time to deadline - Gowalla

Figure 5: Performance comparison of the proposed heuristics when varying the worker set.

spatial crowdsourcing. With participatory sensing, the goal is to exploit the mobile users for a given campaign by leveraging their sensor-equipped mobile devices to collect and share data. Some real-world examples of participatory sensing campaigns include [42], [25] and [35]. For example, the Mobile Millennium project [42] by UC Berkeley is a system that uses GPS-enabled mobile phones to collect en route traffic information and upload it to a server in real time. The server processes the contributed traffic data, estimates future traffic flows and sends traffic suggestions and predictions back to the mobile users. Similar projects were implemented earlier by CarTel [25] and Nericell [35] to collect information about traffic, WiFi access points on the route and road information. All these previous studies on participatory sensing focus on a single campaign and try to address challenges specific to that campaign. However, our focus is on devising a generic crowdsourcing framework, similar to MTurk where multiple campaigns can be handled simultaneously, but spatial. Most existing studies on participatory sensing focus on small campaigns with a limited number of workers, and are not scalable to large spatial crowdsourcing applications. To move from single-campaign and customized participatory-sensing to multi-purpose and generic spatial crowdsourcing, the system needs to scale. That is, it should be able to efficiently assign spatial tasks to workers.

Volunteered Geographic Information: Another class of spatial crowdsourcing is known as volunteered geographic information (or VGI), whose goal is to create geographic information provided voluntarily by individuals. Examples for this class include Google Map Maker [32] and OpenStreetMap [38]. These projects allow the users to generate their own geographic content, and add it to a pre-built map. For example, a user can add the features of a location, or the events occurred at that location. However, the major difference between VGI and spatial crowdsourcing is that in VGI, users voluntarily participate by randomly contributing data, whereas in spatial crowdsourcing, a set of spatial tasks are queried by the requesters, and workers are required to perform those tasks. Moreover, with most VGI projects ([32] and [38]), users are not required to physically present

at the task location/region in order to generate data with respect to that location.

Matching Problems: One can consider the task assignment problem in spatial crowdsourcing as the online bipartite matching problem [26] and [34]. Online bipartite matching problem is relevant to our problem as it captures the dynamism of tasks arriving at different times. However, in online bipartite matching one of the item sets is given in advance and items from the other set arrive (usually one at a time), while in spatial crowdsourcing both sets, i.e., workers and tasks, can come and go without our knowledge. In addition, the performance of an online algorithm is often evaluated based on competitive ratio - the ratio between its performance and the offline algorithm's performance. The online algorithm is competitive if its competitive ratio is bounded under any circumstance; this is not the goal of MC^2 which focuses on the average performance.

Some recent studies in spatial matching [44] and [45] do focus on efficiency and use the spatial features of the objects for more efficient assignment. Spatial matching is a one to one (or in some cases one to many) assignment between objects of two sets where the goal is to optimize over some aggregate (etc., sum, max) function of the distance between matched objects. These studies assume a global knowledge about the locations of all objects exists a priori and the challenge comes from the complexity of spatial matching. However, our problem differs due to the dynamism of tasks and workers (i.e., tasks and workers come and go without our knowledge), thus the challenge is to perform the task assignment at a given instance of time with the goal of global optimization across all times.

Min Set Cover (MSC): MSC has been attracting extensive attention in the research community. Table 2 summarizes the some important discoveries. In [11], a greedy algorithm was proposed, in which at each stage it chooses the set that covers the greatest number of uncovered elements. The greedy algorithm achieves an approximation ratio of $r \leq H(n) \leq ln(n) + 1$, where H is the harmonic series and n is the number of sets. In [31], Lund and Yannakakis showed that set covering cannot be approximated in polynomial time to within a factor of $\frac{1}{2}log_2^n$. Feige [18] improved this bound to $r \geq (1-o(1))ln(n)$ and showed that greedy algorithm is the best-possible polynomial time approximation algorithm for set cover. Other studies [22, 23, 17] improved the approximation ratio to $H(n)-1/6$, $H(n)-1/3$ and $H(n)-1/2$, respectively. In addition, Lan [30] developed an effective heuristic for set cover problem.

Ref	Result
[11]	A greedy algorithm which achieves a lower-bound on approximation ratio $H(n) \leq ln(n)+1$
[31]	An upper-bound of any polynomial-time approximation algorithm $\frac{1}{2}log_2^n$
[18]	Greedy algorithm is the best-possible polynomial time approximation algorithm for set cover $r \geq (1-o(1))ln(n)$
[22]	Improves worst case bound $H(n)-1/6$
[23]	Improves worst case bound $H(n)-1/3$
[17]	Improves worst case bound $H(n)-1/2$
[30]	Effective heuristic for set cover

Table 2: Set Cover Problem

7. CONCLUSION

8. REFERENCES

[1] Amazon mechanical turk. http://www.mturk.com/, 2005.

[2] odesk. https://www.odesk.com/, 2005.

[3] Crowdflower. http://www.crowdflower.com/, 2009.

[4] R. Acharyya, G. K. Das, et al. Unit disk cover problem. *arXiv preprint arXiv:1209.2951*, 2012.

[5] P. K. Agarwal and J. Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Symposium on Computational Geometry*, page 271, 2014.

[6] F. Alt, A. S. Shirazi, A. Schmidt, U. Kramer, and Z. Nawaz. Location-based crowdsourcing: extending crowdsourcing to the real world. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, pages 13–22. ACM, 2010.

[7] V. E. Brimkov, A. Leach, J. Wu, and M. Mastroianni. On the approximability of a geometric set cover problem. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 18, page 19, 2011.

[8] C. Chekuri, K. L. Clarkson, and S. Har-Peled. On the set multicover problem in geometric settings. *ACM Transactions on Algorithms (TALG)*, 9(1):9, 2012.

[9] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 72–83. Springer, 2004.

[10] K.-T. Chen, C.-C. Wu, Y.-C. Chang, and C.-L. Lei. A crowdsourceable qoe evaluation framework for multimedia content. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 491–500. ACM, 2009.

[11] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[12] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.

[13] J. Cranshaw, E. Toch, J. Hong, A. Kittur, and N. Sadeh. Bridging the gap between physical location and online social networks. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages 119–128. ACM, 2010.

[14] H. Dang, T. Nguyen, H. To, and Cyrus. Maximum complex task assignment: Towards tasks correlation in spatial crowdsourcing. 2013.

[15] G. K. Das, R. Fraser, A. Lòpez-Ortiz, and B. G. Nickerson. On the discrete unit disk cover problem. In *WALCOM: Algorithms and Computation*, pages 146–157. Springer, 2011.

[16] G. Demartini, B. Trushkowsky, T. Kraska, and M. J. Franklin. Crowdq: Crowdsourced query understanding. In *CIDR*, 2013.

[17] R.-c. Duh and M. Fürer. Approximation of k-set cover by semi-local optimization. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 256–264. ACM, 1997.

[18] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[19] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information processing letters*, 12(3):133–137, 1981.

[20] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM, 2011.

[21] P. E. from Remotely Sensed Information using Artificial Newral Networks. http://en.wikipedia.org/wiki/PERSIANN, 1997.

[22] O. Goldschmidt, D. S. Hochbaum, and G. Yu. A modified greedy heuristic for the set covering problem with improved worst case bound. *Information Processing Letters*, 48(6):305–310, 1993.

[23] M. M. Halldórsson. Approximating k-set cover and complementary graph coloring. In *Integer Programming and Combinatorial Optimization*, pages 118–131. Springer, 1996.

[24] R. Hassin and A. Levin. A better-than-greedy approximation algorithm for the minimum set cover problem. *SIAM Journal on Computing*, 35(1):189–200, 2005.

[25] 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 4th international conference on Embedded networked sensor systems*, pages 125–138. ACM, 2006.

[26] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. pages 352–258, 1990.

[27] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 189–198. ACM, 2012.

[28] L. Kazemi, C. Shahabi, and L. Chen. Geotrucrowd: Trustworthy query answering with spatial crowdsourcing. 2013.

[29] J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education India, 2006.

[30] G. Lan, G. W. DePuy, and G. E. Whitehouse. An effective and simple heuristic for the set covering problem. *European Journal of Operational Research*, 176(3):1387–1403, 2007.

[31] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

[32] G. map maker. http://www.google.com/mapmaker/, 2008.

[33] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *Proceedings of the VLDB Endowment*, 5(1):13–24, 2011.

[34] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22, 2007.

[35] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[36] N. H. Mustafa and S. Ray. Ptas for geometric hitting set problems via local search. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 17–22. ACM, 2009.

[37] N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

[38] Openstreetmap. http://www.openstreetmap.org/, 2004.

[39] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: Algorithms for filtering data with humans. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 361–372. ACM, 2012.

[40] A. Sorokin and D. Forsyth. Utility data

annotation with amazon mechanical turk. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.

[41] H. To, G. Ghinita, and C. Shahabi. A framework for protecting worker location privacy in spatial crowdsourcing. *Proceedings of the VLDB Endowment*, 7(10), 2014.

[42] UCB. http://traffic.berkeley.edu/, 2008.

[43] J. Whitehill, T.-f. Wu, J. Bergsma, J. R. Movellan, and P. L. Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in neural information processing systems*, pages 2035–2043, 2009.

[44] R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao. On efficient spatial matching. In *Proceedings of the 33rd international conference on Very large data bases*, pages 579–590. VLDB Endowment, 2007.

[45] M. L. Yiu, K. Mouratidis, N. Mamoulis, et al. Capacity constrained assignment in spatial databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 15–28. ACM, 2008.