Real-Time Task Assignment in Hyper-Local Spatial Crowdsourcing under Budget Constraints

Hien To, Liyue Fan, Luan Tran, Cyrus Shahabi
University of Southern California
{hto,liyuefan,luantran,shahabi}@usc.edu

ABSTRACT

Spatial Crowdsourcing (SC) is a novel platform that engages individuals, groups and communities in the act of collecting, analyzing, and disseminating various types of spatial data. This method of data collection can significantly reduce cost and turnover time, and is particularly useful in case of environmental sensing, where traditional means fail to provide fine-grained field data. In this study, we introduce a framework for crowdsourcing hyper-local information, in which only those workers who are already within the spatiotemporal vicinity of a task are eligible candidates to report data, e.g., the precipitation level at their area and time. In this setting, a subset of candidate workers whose size is constrained by a predefined *budget*, can be activated to perform tasks. The challenge is to maximize task coverage under budget constraint, despite the dynamic arrivals of workers and tasks as well as their co-location relationship. We investigate two variants of the problem: fixed and dynamic budgets. For each variant, we study the complexity of its off-line version and then propose effective methods for the online scenario that exploit the spatial and temporal knowledge acquired over time. Extensive experiments with real-world and synthetic datasets show the effectiveness and efficiency of our proposed framework.

1. INTRODUCTION

With the ubiquity of smart phones and wireless network bandwidth improvements, every person with a mobile phone can now act as a multimodal sensor collecting and sharing various types of high-fidelity spatiotemporal data instantaneously. This new paradigm for data collection has various applications in environmental sensing [13], weather forecasting [5], disaster response [19], and transportation decision-making [4]. Specifically, with a few recently developed apps, such as mPING[1], Creek Watch[2] and WeatherSignal[3], individual users can report weather conditions, document wildlife, measure air quality, etc. The gathered data, often in the form of fine-tuned, real-time field reports, offer a valuable addition (e.g., ground-truth data) to the satellite remote sensing and radar detection currently used by various large-scale research projects [6]. The collected data also provide real-time weather notifications to the users who subscribe to the regions of their interest.

[1]http://mping.nssl.noaa.gov/
[2]http://www.research.ibm.com/social/projects_creekwatch.shtml
[3]http://weathersignal.com

Spatial crowdsourcing (SC) [10] offers an effective platform for the aforementioned data collection scenarios where data requesters can create spatial tasks dynamically and workers are assigned with tasks based on their locations and certain optimization objectives, such as distance traveled and maximum assignment [18, 10], trust/reputation [11] and data quality [2]. Consider a scenario where a requester issues a set of rainfall observation tasks to the SC-server (Step 1 in Figure 1), where one task corresponds to a specific geographical region represented by a circle in Figure 1. On the other hand, the workers report and continuously update their locations to the SC-server when they become available for performing tasks (Step 0). Subsequently, the SC-server crowdsources the tasks among the workers in the task region and sends the collected data back to the requester (Steps 2, 3). Apart from precipitation, e.g., rain, snow, drought, drizzle, this framework can be generalized to collect other kinds of information, such as air pollution, light intensity, temperature, noise level, etc.

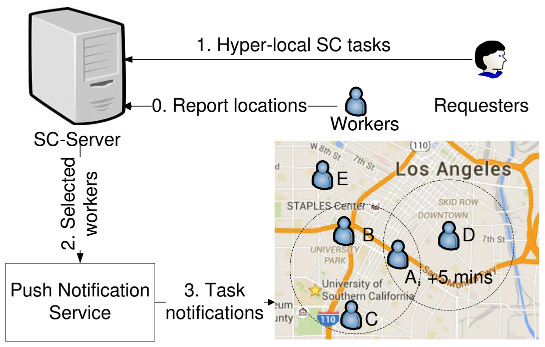


Figure 1: The spatial crowdsourcing framework.

One major difference between our applications and existing SC paradigms [18, 17, 2, 10] is that workers are not required to travel to the task locations, which would encourage participation and yield fast response. We denote this new paradigm as *Hyper-Local Spatial Crowdsourcing.* Furthermore, since the requested data, such as air pollution and temperature, exhibit spatial/temporal continuity in measurement, workers available in close vicinity of the task location and request time are sufficient to fulfill that task. For example, worker B and C in Figure 1 are both eligible to report precipitation level at University of Southern California (USC), and worker A who becomes available 5 minutes later is also qualified. The acceptable ranges of space and time can be specified by data requesters, from which the SC-server can find the set of eligible workers for each task.

In reality, the SC-server operates to maximize fulfilled tasks for revenue. However, it cannot assign every task to all eligible workers due to practical considerations, e.g., to avoid high communication cost for sending/receiving task notifications and worker irritation after receiving too many task notifications. Furthermore, it is not necessary to select many workers for overlapping tasks. For example in Figure 1, the observation of worker A can be used for precipitation tasks at both USC and Los Angeles downtown (tasks shown in two circles).

Apart from considering user reputation [11] for guaranteed data delivery, the goal of our paper is to maximize the number of assigned tasks on the SC-server where only a given number of workers can be selected over a time period, i.e., under a "budget" constraint. When tasks and workers are known *apriori*, we can reduce the task assignment problem to the classic *Maximum Coverage Problem* and its variants. However, the main challenge with SC comes from the dynamism of the arriving tasks and workers, which renders an optimal solution infeasible in the real-time scenario. In Figure 1, the SC-server is likely to activate worker D and either worker B or C for the two tasks, respectively, without knowing that a more favorable worker A is qualified for both tasks and will arrive in the near future. Previously developed heuristics in literature [18, 17, 2, 10] do not consider the vicinity of tasks in space and time or the budget, thus cannot be applied to Hyper-Local SC.

Due to the unforeseeable arrival time and location of tasks and workers, the dynamic allocation of the budget towards global optimization imposes another challenge for real-time applications. As one worker can be selected for tasks from different time periods (e.g., worker A in Figure 1), greedily optimizing within each time period does not guarantee global optimal solutions over the entire campaign. We consider two variations for budget allocation, i.e., given a budget for each time period or given a budget for the entire campaign, which enforce a constraint on the number of activated workers for each time period or for the entire campaign, respectively. The specific contributions of this paper are as follow.

1) We provide a formal definition of the Hyper-Local SC problem, where the goal is to maximize task coverage under budget constraints. To the best of our knowledge, we are the first to study the problem. Two problem variants are investigated in the paper: one with a given budget for each time period, and the other with a given budget for the entire campaign. We study the problem complexity of both variants (assuming a clairvoyant server) and prove that they are NP-hard in the offline setting.

2) When a budget is specified for each time period, we propose three heuristics in the online setting, i.e., *Basic*, *Spatial* and *Temporal*. The spatial heuristic favors the tasks that may not co-locate with the workers who will be available in the near future, while the temporal heuristic favors the tasks which will soon be expired.

3) When a budget is specified for an entire campaign, we devise an adaptive strategy based on the contextual bandit algorithm [12] to dynamically allocate budget over time. By utilizing the historical logs and introducing randomness, our strategy strikes a balance between *exploitation* and *exploration* and captures the dynamic changes in the arriving patterns of workers and tasks.

4) We conduct extensive experiments with real-world and synthetic datasets. The empirical results confirm that our online solutions are efficient and increase the *task coverage* by 40% over the baseline approach, thus can enable a wide range of SC applications.

The remainder of this paper is organized as follows. Section 2 shows the related work. Section 3 presents the preliminaries of our SC framework and defines notations for the hyper-local SC problem. In Section 4, we introduce the problem variant with a fixed budget for each time period and present heuristics for the online case. In Section 5, we study the problem variant where a budget is given for the entire campaign and introduce adaptive budget allocation strategies. Section 6 reports our experimental results while Section 7 concludes the paper and proposes future directions.

2. RELATED WORK

Spatial Crowdsourcing (SC) has recently been attracting attention in both the research communities (e.g., [2, 17, 10]) and industry (e.g., TaskRabbit, Gigwalk [14]). A recent survey in this area can be found in [18], which distinguishes SC from related fields, including crowdsourcing, participatory sensing, volunteered geographic information. In [10], a SC framework whose goal is to maximize the number of assigned tasks is proposed. In [17], the authors introduce the problem of protecting worker location privacy in SC. This study proposes a framework that achieves differentially-private protection guarantees. Recently in [2], Cheng et. al. study reliable task assignment in SC whose objective is to maximize both the confidence of task completion and the diversity quality of the tasks. In contrast to our study, in [2, 17, 10], the workers need to travel to the tasks' locations, which may take a long time in rush hour. Consequently, the workers may reject their assigned tasks. One of the challenges with crowdsourcing is to aggregate the workers' responses with varying degree of accuracy. One of the well-known mechanisms is majority voting, which accepts the result supported by the majority of workers. Another work aims to tackle the issue of trust by having tasks performed redundantly by multiple workers [2]. In the scope of this study, we assume that selected workers provide trustworthy data. If there are multiple reports for one task the SC-server will send all the corresponding results to the task requester.

Crowdsourcing hyper-local information has been presented in both research (e.g., mPING [6]) and industry (e.g., mPing[4], WeatherSignal[5]). mPing is the state-of-the-art system for crowdsourcing weather reports; anyone with a smartphone can report precipitation observations. However, mPing neglects to consider the cost of selecting workers to perform tasks. Also, the task assignment in mPing is often suboptimal as workers do not have a global system view. Workers typically choose nearby tasks to them, which may cause multiple workers to cover the same task *unnecessarily* while many other tasks remain uncovered. WeatherSignal uses phone sensors, such as temperature and humidity sensors, to measure local atmospheric conditions. Particularly, an algorithm was employed to translate phone battery temperature into the ambient temperature. However, the algorithm accuracy is the main concern. In addition, WeatherSignal is restricted to the availability of phone sensors, which currently cannot detect various kinds of hyper-local information, such as air pollution, noise level, light intensity and precipitation level such as rain, snow, drought. Our proposed framework overcomes the aforementioned shortcomings by leveraging human power and server-side optimization of task coverage.

3. PRELIMINARIES

We first introduce concepts and notations used in this paper.

A **task** is a query of certain hyper-local information, e.g., precipitation level at a particular location and time. For simplicity, we assume that the result of a task is in the form of a numerical value, e.g., *0=rain,1=snow,2=none*[6]. Specifically, every task comes with a pre-defined region where any enclosed user can report data for that task. In this paper, we define each task region as a circular space centered at the task location; however, task region can be extended to other shapes such as polygon or to represent geography such as district, city, county, etc. In real applications, the proper radius of the task region can be set by the requester. Moreover, each task also specifies a valid *time interval* during which users can provide data. More formally,

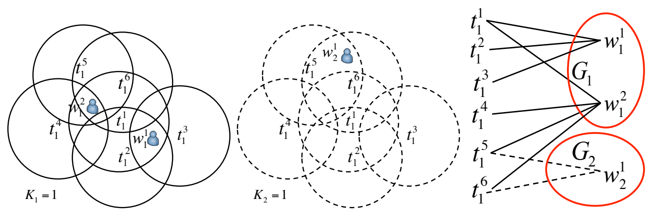
Definition 1 (Task). *A task t of form $<l, r, s, \delta>$ is a query at location l, which can be answered by workers within a circular space centered at l with radius r. The parameter δ indicates the duration of the query: it is requested at time s and can be answered until time $s + \delta$.*

We refer to $s + \delta$ as the "deadline" of task t. A task expires if it has not been answered before its deadline. Figure 2a shows the regions of six tasks, $t_1^1, t_1^2, ..., t_1^6$. All tasks expire at time period 2 (i.e., they can be deferred to time period 2), represented by the dashed circles in Figure 2b. A **worker** can accept task assignments when he is *online*.

Definition 2 (Worker). *A worker w of form $<id, l>$, is a carrier of a mobile device who can accept spatial task assignments. The worker can be uniquely identified by his id and his location is at l.*

Intuitively, a worker is eligible to perform a task if his location is enclosed in the task region. In Figure 2a, w_1^1 is

[4]Developed by National Severe Storms Laboratory (NSSL)
[5]weathersignal.com
[6]Remote sensing techniques based on satellite images cannot differentiate between rain and snow



(a) Time period 1 (b) Time period 2 (c) Bipartite graph

Figure 2: Graphical example of worker-task coverage ($\delta = 2$). Subscripts represent time periods while superscripts mean ids.

eligible to perform t_1^1, t_1^2 and t_1^3 while w_1^2 is qualified to perform t_1^1, t_1^4, t_1^5 and t_1^6. Furthermore, a worker's report to one task can also be used for all other unexpired tasks whose task regions enclose the worker. As in Figure 2b, online worker w_2^1 is eligible to perform t_1^5 and t_1^6, which are deferred from time 1.

Let $W_i = \{w_i^1, w_i^2, ...\}$ denotes the set of available workers at time s_i and $T_i = \{t_i^1, t_i^2, ...\}$ denotes the set of available tasks including tasks issued at time s_i and previously issued unexpired tasks. Below we define the notions of **worker-task coverage** and **coverage instance sets**.

Definition 3 (Worker-Task Coverage). *Given $w_i^j \in W_i$, let $C(w_i^j) \subset T_i$ denotes the task coverage set of w_i^j, such that for every $t_i^k \in C(w^j)$,*

$$s_i < t_i^k.(s + \delta) \quad (1)$$

$$||w_i^j.l - t_i^k.l||_2 \leq t_i^k.r \quad (2)$$

We also say the worker w_i^j covers the tasks $t_i^k \in C(w_i^j)$. An example of a coverage in Figure 2a is $C(w_1^1) = \{t_1^1, t_1^2, t_1^3\}$.

Definition 4 (Coverage Instance Set). *At time s_i, the coverage instance set, denoted by I_i is the set of worker-task coverage of form $<w_i^j, C(w_i^j)>$ for all workers $w_i^j \in W_i$.*

Time	Coverage Instance Sets
1	$\{(w_1^1,<t_1^1, t_1^2, t_1^3>), (w_1^2,<t_1^1, t_1^4, t_1^5, t_1^6>)\}$
2	$\{(w_2^1,<t_1^5, t_1^6>)\}$

Table 1: The coverage instance set of the example in Figure 2.

The coverage instance sets for the example in Figure 2 are illustrated in Table 1. For simplicity, we first assume the utility of a specific task assignment is *binary* within the task region and before the deadline. That is, assignment to any worker within a task region before the deadline has utility 1, i.e. 1 successful assignment, and 0 otherwise. As a result, task t_1^5 and t_1^6 being answered by worker w_1^2 at time 1 is equivalent to it being answered by w_2^1 at time 2.

The goal of our study is to maximize task assignment given a budget, despite the dynamic arrivals of tasks and workers. Now, we formally define the notion of a budget.

Definition 5 (Budget). *Budget K is the maximum number of workers to select in a coverage instance set.*

In practice, budget K can capture the *communication cost* the SC-server incurs to push notifications to K-selected workers (Step 3 in Figure 1), or the *rewards* paid to the K workers who provide data.

4. FIXED BUDGET

We investigate the first variant of the maximum task coverage problem, in which a fixed budget is given for each time period. We first study the problem complexity of the off-line case and then propose heuristics for the online setting.

4.1 Offline Scenario

Problem 1 (Fixed-budget Maximum Task Coverage). *Given a set of time periods $\phi = \{s_1, s_2, ..., s_Q\}$ and a budget K_i for each s_i, the fixed-budget maximum task coverage (MTC) problem is to select a set of workers L_i at s_i, such that the total number of covered tasks $|\bigcup_{i=1}^{Q} \bigcup_{w_i^j \in L_i} C(w_i^j)|$ is maximized and $|L_i| \leq K_i$.*

This optimization problem is challenging since each worker is eligible for a subset of tasks. The fact that a task can be deferred to future time periods further adds to the complexity of the problem. With the following theorem, we proof that *fixed-budget offline MTC* is NP-hard by a reduction from the *maximum coverage with group budgets* constraints problem (MCG) [1]. MCG is motivated by the maximum coverage problem (MCP) [7]. Consider a given I_g, we are given the subsets $S = \{S_1, S_2, ...S_m\}$ of a ground set X and the disjoint sets $\{G_1, G_2, ..., G_l\}$. Each G_i, namely a *group*, is a subset of $S = \{S_1, S_2, ...S_m\}$. With MCG, we are given an integer k, and an integer bound k_i for each group G_i. A solution to I_g is a subset $H \subset S$ such that $|H| \leq k$ and $|H \cap G_i| \leq k_i$ for $1 \leq i \leq l$. The objective is to find a solution such that the number of elements of X covered by the sets in H is maximized. MCP is the special case of MCG [1]. Since MCP is known to be strongly NP-hard [7], by restriction, MCG is also NP-hard.

Theorem 1. *Fixed-budget offline MTC is NP-hard.*

Proof. We prove the theorem by a reduction from MCG [1]. That is, given an instance of the MCG problem, denoted by I_g, there exists an instance of the MTC problem[7], denoted by I_t, such that the solution to I_t can be converted to the solution of I_g in polynomial time. The reduction has two phases, *transforming* all workers/tasks across the entire campaign to a bipartite graph, and *mapping* from MCG to MTC. First, we layout the tasks and workers as two set of vertices in a bipartite graph in Figure 2c. A worker w_i^j can cover a task t_i^k if both spatial and temporal constraints hold, i.e., Equations 2 and 1, respectively. In Figure 2c, w_2^1 can cover t_1^4 and t_2^5, which are deferred from s_1 to s_2, represented by the dashed line.

Thereafter, MTC can be stated as follows. Selecting the maximum K_i workers per group, each group represents a time period, such that the number of covered tasks is maximized (i.e., $|L_i| \leq K_i$). To reduce I_g to I_t, we show a mapping from I_g components to I_t components. For every elements in the ground set X in I_g, we create a task t_i^j $(1 \leq j \leq |X|)$. Also, for every set in S, we create a worker w_i^j with $C(w_i^j = S_j)$ $(1 \leq j \leq m)$. Consequently, to solve I_t, we need to find a subset $L_i \subset W_i$ workers of maximum size K_i in each group whose coverage is maximized. Clearly, if an answer to I_t is the set L_i $(1 \leq i \leq Q)$, the answer to I_g will be the set $H \subset S$ of maximum coverage such that $|H| \leq k = \sum_{i=1}^{Q} K_i$ and $|H \cap G_i| \leq k_i = K_i$ for $1 \leq i \leq Q$.

As the transformation is bounded by the polynomial time to construct the bipartite graph, this completes the proof. □

[7]In this section, MTC refers to fixed-budget MTC for short

By a reduction from the MCG problem, we can now use any algorithm that computes MCG to solve the MTC problem. The solution to the example in Figure 2c is $\{w_1^1, w_2^1\}$. It has been shown in [1] that the greedy algorithm gives 0.5-approximation for MCG and it is tight. However, this solution is not feasible in the online scenario where the server does not have knowledge about future tasks/workers.

4.2 Online Scenario

The main challenges of spatial crowdsourcing are due to the dynamic arrivals of workers and tasks. The SC-server must select workers frequently and in real-time as new tasks and workers become available or as tasks are completed (or expired) and workers leave the system. Since the clairvoyant assumption is not practical, the SC-server tries to optimize the worker selection locally at every period of time. However, the optimization problem within each time period is not straightforward either. In fact, fixed-budget online MTC is NP-hard by restriction. MTC for a single time snapshot $(Q = 1)$ is NP-hard; thus, fixed-budget online MTC is also NP-hard. MTC for one-time snapshot can be reduced from the maximum coverage problem (MCP) [7], MCP $\leq_p$ MTC one-time snapshot.

As MCP is strongly NP-hard, a greedy algorithm is proposed to achieve an approximation ratio of 0.63 [7]. The algorithm chooses a set at each stage that contains the largest number of *uncovered* elements. The results in [7] show that the greedy algorithm is the best-possible polynomial time approximation algorithm for MCP. As mentioned earlier, a global solution to online MTC is not feasible. Hence, our approach is to solve the MTC problem for every period of time with several heuristics, namely *Basic*, *Spatial* and *Temporal*.

4.2.1 Basic Heuristic

The *basic* approach solves the online MTC problem by using the greedy heuristic [8] for every time period. At each stage, *Basic* selects the worker that covers the maximum number of *uncovered* tasks, depicted in Lines 9-10 of Algorithm 1. For instance, in Figure 2a, w_1^1 is selected at the first stage. At the beginning of each time period, Line 4 removes expired tasks being carried from the previous time period (Line 13). Thereafter, Line 5 adds uncovered unexpired tasks to current task set and assign new indices to tasks in U'_{i-1}. Line 12 outputs the covered tasks C_i per time period that will be used as the main performance metric in Section 6. The algorithm terminates when either running out of budget or all the tasks are covered (Line 9).

4.2.2 Spatial Heuristic

The *basic* strategy treats all tasks equally without considering the spatial characteristic of task locations. However, a task located in an area popular with workers is more likely to be covered in the future and vice versa. Therefore, tasks located in worker-sparse areas should have higher priorities to be assigned. Consequently, the priority of a worker is high if he covers a larger number of high priority tasks.

Algorithm 1 Basic Algorithm

```
1: Input: worker set $W_i$, task set $T_i$, budgets $K_i$
2: Output: selected workers $L_i$
3: For each time period $s_i$
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 enclose any worker $T'_i \leftarrow T_i$
7:     Construct worker set $W_i$, each $w_i^j$ contains $C(w_i^j)$
8:     Init $L_i = \{\}$, uncovered tasks $R \leftarrow \bigcup_{w_i \in W_i} C(w_i^j)$
9:     While $|L_i| < K_i$ and $|R| > 0$
10:        Select $w_i^j \in W_i - L_i$ that maximize $|C(w_i^j) \cap R|$
11:        $R \leftarrow R - C(w_i^j)$; $L_i \leftarrow L_i + w_i^j$
12:    $C_i \leftarrow \bigcup_{w_i^j \in L_i} C(w_i^j)$
13:    Keep uncovered tasks $U_i \leftarrow T'_i - C_i$
```

The spatial "popularity" of a task location can be measured using Location Entropy [3], which captures the diversity of visits to a location. A location has a high entropy if many workers visit that location with equal probabilities. In contrast, a location has a low entropy if there are only a few workers visiting. We extend the concept of location entropy to *region entropy* of the task.

For a given task t, let O_t be the set of visits to task region $R(t.l, r)$. Also, let W_t be the set of distinct workers that visited the task region of t, and $O_{w,t}$ be the set of visits that worker w has made to the location of task t. The probability that a random draw from O_t belongs to $O_{w,t}$ is $P_t(w) = \frac{|O_{w,t}|}{|O_t|}$, which is the fraction of total visits to t that belong to worker w. The region entropy for t is computed as follows.

$$RE(t) = -\sum_{w \in W_t} P_t(w) \times log P_t(w) \quad (3)$$

We discretize the space using a 2D grid and pre-compute the entropies for each grid cell using aggregated historical data. By computing the region entropy of every *uncovered* task of a worker w_i^j, we associate to every worker a priority.

$$priority(w_i^j) = \sum_{t_i^k \in C(w_i^j) \cap R} \frac{1}{1 + RE(t_i^k)} \quad (4)$$

where $RE(t_i^k)$ is calculated as Equation 3. Consequently, *Spatial* greedily selects the worker with maximum *priority* at each stage, as opposed to the worker that covers the maximum number of uncovered tasks. Line 10 in Algorithm 1 can be modified to reflect the spatial priority of each worker.

4.2.3 Temporal Heuristic

Another approach to prioritizing tasks is by their temporal urgency. The intuition is that a task that is further away from its deadline is more likely to be covered in the future, and vice versa. As a result, near-deadline tasks are more important than others and a worker that covers a large number of soon-to-expire tasks should be preferred for selection. Similar to Equation 4, the priority of a worker w_i^j is defined based on the time-to-deadline for each of his covered tasks.

$$priority(w_i^j) = \sum_{t_i^k \in C(w_i^j) \cap R} \frac{1}{t_i^k.(s + \delta) - i} \quad (5)$$

Likewise, Line 10 in Algorithm 1 can be updated with the temporal priority for each worker as in Equation 5. We will empirically evaluate all heuristics in Section 6.

5. DYNAMIC BUDGET

Another problem variant we investigate in the paper is more general, where a budget is given over the entire campaign. This scenario often results in higher task coverage. In Figure 2, the fixed-budget method selects w_1^1 and w_2^1 and obtains the coverage of 5. However, the dynamic-budget variant yields higher coverage of 6 by selecting w_1^1 and w_1^2. We study the problem complexity in the offline case and propose a framework for adaptive budget allocation strategies.

5.1 Offline Scenario

Problem 2 (Dynamic-budget Maximum Task Coverage). *The dynamic-budget MTC problem is similar to Problem 1, except the total budget is specified for the entire campaign K rather than for each time period, i.e., $\sum_{i=1}^{Q} |L_i| \le K$.*

We first investigate the offline case, where the server is clairvoyant about the future workers and tasks. By restriction, we prove dynamic-budget offline *MTC* is NP-hard by reduction from the maximum coverage problem (MCP) [7]. The reduction includes two steps, *transformation* and *mapping*. The first step is similar to that of Theorem 1, in which the workers and tasks from the entire campaign are transformed into a bipartite graph as illustrated in Figure 2c. The *mapping* step can be considered as a special case of the proof in Theorem 1, in which there exists only one group of all budget.

5.2 Online Scenario

We propose adaptive strategies to dynamically allocate the total budget K over Q time periods ($K \ge Q$). When a budget is spent during a particular time period, the previously proposed heuristics, i.e., *Basic, Spatial, Temporal*, can be applied.

5.2.1 Adaptive Budget Allocation

The simplest strategy, namely *Equal*, equally divides K to Q time periods; each has K/Q budget and the last time period obtains the remainder. However, *Equal* may over-allocate budget to the time periods with small numbers of tasks. Another strategy is to give each time period a budget that is proportional to the number of available tasks at that time period $|T_i|$ (i.e., $\propto \frac{|T_i|}{|T|} K$), where $|T|$ is the total number of tasks. However, $|T|$ is unknown *a priori* and $|T_i|$ does not provide insight on whether they can be optimally covered by available workers.

To maximize task assignment, we need an approach that adaptively adjust the budget for each time period according to the dynamic arrivals of tasks and workers and their co-location. Toward that end, what is the indication of budget over/under-utilization? And how do we adapt to the changing coverage instance sets over time? To capture the aforementioned two aspects, we define the following two notions. **Delta budget**, denoted as δ_K, captures the current status of budget utilization, compared to a baseline budget strategy. Given a certain baseline $\{K^{base}[i], i = 1, \ldots, Q\}$,

δ_K is the difference between the aggregated baseline budget up to time i and the budget used so far.

$$\delta_K = \sum_{t=1}^{i} (K^{base}[t]) - K_{used} \tag{6}$$

We adopt the *Equal* strategy as a baseline. Another notion denoted as δ_λ is **delta gain**, which represents the importance of a worker being considered (λ_i) compared to the ones selected in the past ($\overline{\lambda_{i-1}}$). Formally, at time i,

$$\delta_\lambda = \lambda_i - \overline{\lambda_{i-1}} \tag{7}$$

where λ_i is the current gain, calculated from our heuristics (i.e., as $|priority(w_i^j)|$ with *spatial* and *temporal* heuristics) and $\overline{\lambda_{i-1}}$ is the average gain of the *last* added workers from the previous time periods, $\overline{\lambda_{i-1}} = \frac{1}{j-1} \sum_{t=1}^{j-1} \lambda_t$.

Based on the contextual information δ_K and δ_λ at each time period, we examine available workers one by one using a particular heuristic and decide whether to allocate budget 1 for each worker. Intuitively, when both δ_K and δ_λ are positive, i.e., the budget is under-utilized and a worker has higher gain than the historical average, the selection of the considered worker is favored. On the other hand, when both are negative, it is not worthwhile to spend the budget. The cases where one is positive and the other is negative are hard to judge as we prefer workers with higher gains but would also like to save budget for the future time periods.

Our solution to the sequential decision problem is inspired by the well-know multi-armed bandit problem (MAB) [16], which has been widely studied and applied to decisions in clinical trials [15], online news recommendation [12], and portfolio design [9]. ϵ-greedy, which achieves a trade-off between exploitation and exploration, proves to be often hard to beat by other MAB algorithms [20]. Hence, we propose an adaptive budget allocation, based on *contextual ϵ-greedy* algorithm [12]. We illustrate our solution in Figure 3.

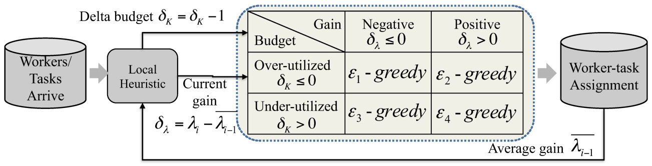

Delta budget $\delta_K = \delta_K - 1$
Workers/ Tasks Arrive
Local Heuristic
Current gain
$\delta_\lambda = \lambda_i - \overline{\lambda_{i-1}}$
Gain
Budget
Negative $\delta_\lambda \le 0$
Positive $\delta_\lambda > 0$
Over-utilized $\delta_K \le 0$
Under-utilized $\delta_K > 0$
ε_1 - greedy
ε_2 - greedy
ε_3 - greedy
ε_4 - greedy
Worker-task Assignment
Average gain $\overline{\lambda_{i-1}}$

Figure 3: Work-flow for adaptive budget allocation.

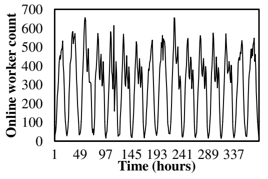
Contextual ϵ-greedy for Adaptive Budget Allocation. For each worker selected by our heuristics, a binary decision to make is whether to allocate budget 1 to activate that worker. A YES decision corresponds to *exploration*, as the action will update our prior knowledge about gains of the selected workers while a NO decision corresponds to *exploitation*. Intuitively, the best decision varies according to the circumstances of each worker. The contextual ϵ-greedy algorithm allows us to specify an exploration-exploitation rate based on the worker's context, i.e., δ_K and δ_λ. As depicted in Figure 3, an ϵ_i-greedy algorithm is activated corresponding to the context information, in which a YES decision is made with $1 - \epsilon_i$ probability and ϵ_i probability for a NO decision. The outline of our adaptive strategy is depicted in Algorithm 2. By default, $\epsilon_2 = \epsilon_3 = 0.5$ (uniform), while the YES/NO decisions are clearly made in the other cases ($\epsilon_1 = 1, \epsilon_4 = 0$).

Algorithm 2 Adaptive Budget Algorithm (Adapt)

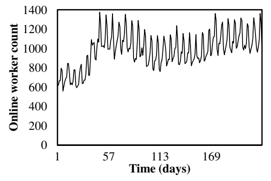
```
1:  Input: $W_i$, $T_i$, total budgets $K$
2:  Output: selected workers $L_i$
3:  Init $R = T_i$; used budget $K_{used} = 0$; average gain $\overline{\lambda_{i-1}} = 0$
4:  Budget allocation $K^{equal}[]$ with Equal strategy
5:  For each time period $s_i$
6:      Perform Lines 4-8 from Algorithm 1
7:      Remained budget $K_i = K - K_{used}$
8:      If $i = Q$, then $\delta_K = K_i$ {the last time period}
9:      Otherwise, $\delta_K = (\sum_{t=1}^{i} K^{equal}[t]) - K_{used}$
10:     While $|L_i| < K_i$ and $R$ is not empty:
11:         Select $w_i$ in $W_i$ with highest $\lambda_i$
12:         Delta gain $\delta_\lambda = \lambda_i - \overline{\lambda_{i-1}}$
13:         If $\delta_\lambda \le 0$ and $\delta_K \le 0$ and $rand(0,1) \le \epsilon_1$, then break
14:         If $\delta_\lambda \le 0$ and $\delta_K > 0$ and $rand(0,1) \le \epsilon_2$, then break
15:         If $\delta_\lambda > 0$ and $\delta_K > 0$ and $rand(0,1) \le \epsilon_3$, then break
16:         If $\delta_\lambda > 0$ and $\delta_K \le 0$ and $rand(0,1) \le \epsilon_4$, then break
17:         $\delta_K = \delta_K - 1$
18:         Perform Line 11 from Algorithm 1
19:     $K_{used} = K_{used} + |L_i|$ {update the budget}
20:     $\overline{\lambda_i} = (\overline{\lambda_{i-1}}(Q-1) + \lambda_i)/Q$
21:     Perform Lines 12,13 from Algorithm 1
```

5.2.2 Historical Workload

Algorithm 2 has been simplified by considering *Equal* as the baseline budget. Therefore, we propose to use the historical data to compute a baseline that captures the worker/task arrival patterns. Intuitively, human activity exhibits temporal patterns and understanding those patterns can predict the availability of workers. Musthag et al. [14] show the time-of-day usage pattern of workers in real mobile crowdsourcing applications. The activity peaks are between 4 to 7 pm when workers leave their day jobs. Similar patterns are observed in the data sets in Figure 4, which will be introduced in Section 6.1. Figure 4a shows the number of check-ins per hour in the Foursquare dataset. During weekdays, activity presents three peaks: in the morning when people go to work, at lunchtime, and between 6 pm and 8 pm when they commute. During weekends, user activity presents one long-lasting plateau between noon and 10 pm. As for weekly patterns, we can observe peak activity during weekends in Gowalla dataset in Figure 4b.



(a) Foursquare, 16x24 hours



(b) Gowalla, 32x7 days

Figure 4: The number of check-ins vary over time periods.

As workers have activity patterns, we further improve Algorithm 2 by leveraging the optimal budget strategy in the recent past and use that as the baseline in Equation 6. The idea is to learn the budget allocation of from historical data (i.e., on day 1), namely *workload*, using the solution to the dynamic-budget offline *MTC* problem and use the results to guide the future time periods (i.e., on day 2). Consequently, we propose to improve Algorithm 2 by replacing Line 4 with the most recent optimal budget allocation, $K^{prev}[]$.

6. PERFORMANCE EVALUATION

Name	#Tasks	#Workers	MTD [8]	$|s_i|$
Foursquare	89,968	45,138 ($90/km^2$)	16.6km	1 hour
Gowalla	151,075	6,160 ($35/km^2$)	3.6km	1 day

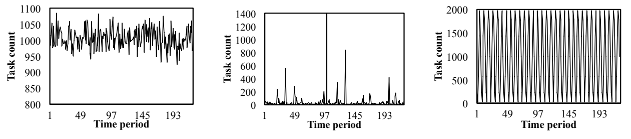
Table 2: Characteristics of real data.

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

6.1 Experimental Methodology

We used real-world data from location-based applications (Gowalla and Foursquare) that have been used in [18, 17, 10] to emulate spatial crowdsourcing (SC) workers and tasks, summarized in Table 2. Gowalla contains the check-in history of users in a location-based social network. For our experiments, we used the check-in data over a period of 224 days in 2010, including more than 100,000 spots (e.g., restaurants), covering the state of California. We assumed that Gowalla users are SC workers, and their locations are those of the most recent check-in points. We also picked the granularity of a time period as one day. Consequently, all the users who checked in during a day as our available workers for that time period. At each time period, 1000 spatial tasks were randomly selected from the Gowalla venues. Likewise, Foursquare contains the check-in history of 45,138 users to 89,968 venues over 384 hours in the area of Pittsburgh, Pennsylvania. Similarly, we used Foursquare users as SC workers and venue locations as tasks. Last but not least, we developed a spatial crowdsourcing system namely *iRain* to collect weather information. Since the iRain dataset is relatively small, we generated a synthetic dataset with similar spatial distributions of workers and tasks (i.e., 1,355 workers and 385 tasks). The entire space is discretized into 200x200 grid, which are in forms of 2D histograms. For each time instance, we randomly create workers and tasks into grid cells proportional to their density, the locations are randomly distributed within each cell.

We generated a range of datasets by combining the spatial worker/task distributions and their arrival rate. We needed to generate only task counts as the worker counts can be obtained from Gowalla (mean=991) and Foursquare (mean=291). To generate the number of available tasks (mean = 1000) for every time period, we used four functions: CONST, POISSON (default), ZIPFIAN and COSINE (Figure 5). For example, Go-POISSON uses Gowalla for the spatial distributions and the worker arrival rate (Fig. 4b) and POISSON for the task arrival rate (Fig. 5a). iRain-POISSON uses iRain for the spatial distributions and POISSON for the arrival rates of workers and tasks.



(a) POISSON (b) ZIPFIAN (c) COSINE

Figure 5: Three samples of synthetic workload data.

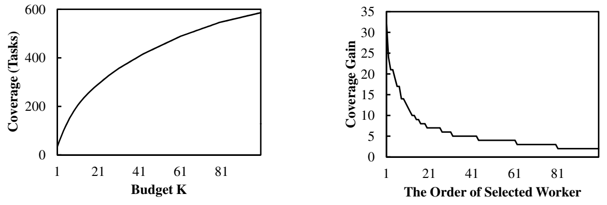
In all of our experiments, we varied the number of time periods $Q \in \{7, 14, \mathbf{28}, 56\}$ and the task duration $\delta \in \{1, 2, 3, 4, \mathbf{5}, 6, 7, 8, 9, 10\}$. We varied the budget $K \in \{28, \mathbf{56}, ..., 3556\}$ and the task radius $r \in \{1, 2, 3, 4, \mathbf{5}, 6, 7, 8, 9, 10\}$ km. For Foursquare, $Q \in \{\mathbf{24}, 48, 72, 96\}$ and $K \in \{24, \mathbf{48}, ..., 3048\}$ because we modeled a time period as one hour. Default values are shown in boldface. For Zipfian decrease function, skew parameter s is set to 1. In fixed-budget experiments, we set a budget for each time period to K/Q. All measured results are aggregated over $224/Q$ runs for Gowalla, and $384/Q$ runs for Foursquare.

6.2 Experimental Results

We evaluate our solutions in terms of *task coverage* and *relative improvement* measured by the coverage difference divided by the coverage of the baseline approach.

6.2.1 *Offline Solutions*

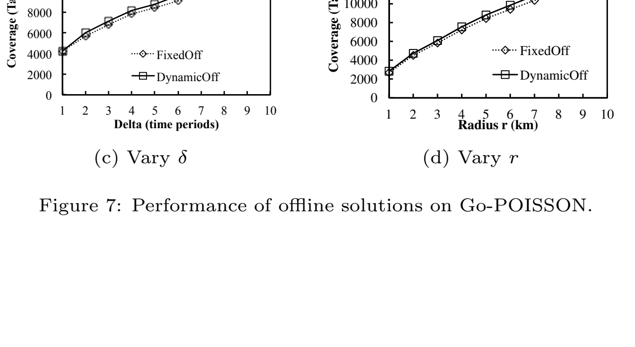
Monotonicity and Diminishing Return: We discuss two interesting properties of the solutions by *Basic* algorithm to the one-time snapshot *MTC* problem (i.e., $Q = 1$). We observe that the number of covered tasks monotonically increases with plurality budget K as illustrated in Figure 6a. Also, Figure 6b shows the heavy-tailed distribution of the coverage gain of workers selected at each stage of *Basic*.



(a) Monotonicity increase (b) Diminishing return

Figure 6: Solutions of *Basic* for *MTC* instance, Gowalla.

FixedOff v.s. DynamicOff: We compare the offline solutions to the two problem variants, *FixedOff* (Section 4.1) and *DynamicOff* (Section 5.1), using the greedy algorithm. Figure 7a illustrates the results for Go-POISSON by varying the budget. As expected, higher budget yields higher coverage. However, the higher the budget, the smaller the *relative* improvement as shown in Figure 7b. This effect can be explained by the diminishing return property. That is, the coverage differences among the algorithms are small in the case of high budget.

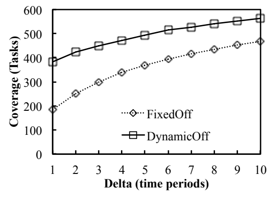


(a) Vary K (b) Vary K (c) Vary δ (d) Vary r

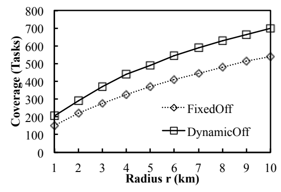
Figure 7: Performance of offline solutions on Go-POISSON.

We also evaluate the offline solutions by varying task duration δ. As expected, Figure 7c shows that longer δ results in higher coverage. Also, the improvement of *DynamicOff* over *FixedOff* is larger when δ increases. The reason is that when tasks can be deferred to a wider range of future time periods, dynamic budget allocation becomes more effective. In Figure 7d, when r increases, every task can be covered by more workers, which yield higher coverage.

We did not observe much difference between fixed budget and dynamic budget for Go-POISSON since the worker/task arrivals are stable. However, when there are peaks in arrival rate, such as in Go-ZIPFIAN, *DynamicOff* shows more advantage over *FixedOff* (by up to 110% at $\delta = 1$ in Figure 8). Unlike the result in Figure 7c, Figure 8a shows large improvements at $\delta = 1$. The reason is that under the spiky workload, *FixedOff* uses a fixed amount of budget to the time periods with high spikes while *DynamicOff* can allocates more budget to those time periods to cover more tasks.



(a) Vary δ

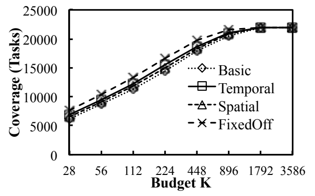


(b) Vary r

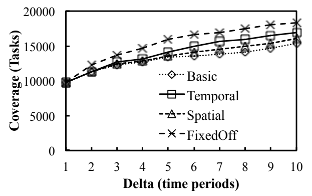
Figure 8: Performance of offline solutions on Go-ZIPFIAN.

6.2.2 Online Solutions

The Performance of Heuristics: We evaluate the performance of the online heuristics from Section 4, *Basic*, *Spatial* and *Temporal*. Figures 9a shows the relative improvements of *Spatial* and *Temporal* over *Basic* on Go-POISSON. *Spatial* and *Temporal* yield 12% and 5% higher coverage than *Basic* at $K = 28$ and their performance converges as K increases. In addition, Figure 9b shows the results by varying task duration δ. As expected, the improvements of *Spatial* and *Temporal* are higher at larger δ while all techniques perform similar at $\delta = 1$. Similar trends can be observed when increasing the task radius r. Due to the superior performance, we will adopt *Temporal* from now on.



(a) Vary K, Go-POISSON



(b) Vary δ, Go-POISSON

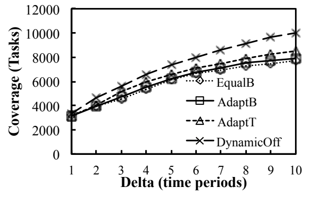
Figure 9: Performance of heuristics in the fixed-budget scenario.

Adaptive Budget Strategy: We evaluate the performance of the adaptive algorithms in Section 5.2.1. *EqualB* refers to the algorithm that divides the budget equally to time periods and runs *Basic*, whereas *AdaptB* and *AdaptT* use Algorithm 2 with *Basic* and *Temporal*, respectively. Figures 10a and 10b show the improvements of *AdaptT* over *AdaptB* and *EqualB* by varying task duration δ. As expected, the higher δ, the larger improvements. Particularly, *AdaptT* improves *EqualB* by up to 12%. As *AdaptT* improves *AdaptB*,

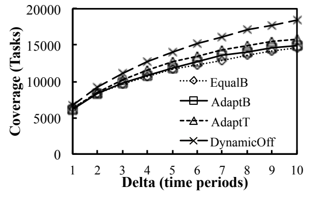
Radius	EqualB	AdaptB	AdaptT
1	8932	9146	9396
5	24819	25102	25321
10	26859	27142	27442
Delta			
1	18564	18716	19251
5	24620	24991	25112
10	24819	25156	25274

Table 3: The coverage of *AdaptT* on iRain-POISSON. $K = 56, Q = 28$.

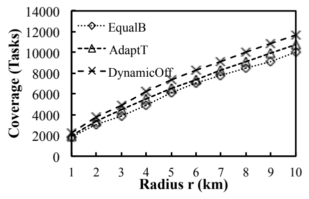
we hereafter show only the results of *AdaptT*. Figures 10c and 10d show similar results by varying task radius.



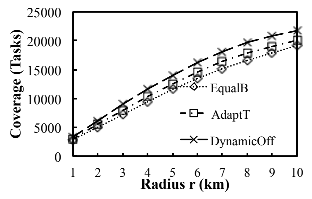
(a) Vary δ, Go-POISSON



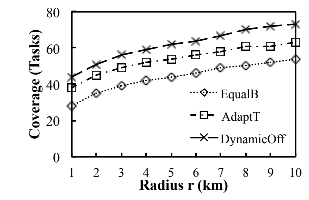
(b) Vary δ, Fo-POISSON



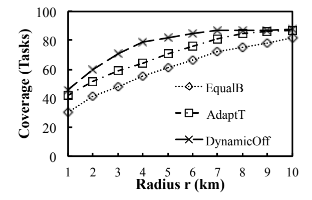
(c) Vary r, Go-POISSON



(d) Vary r, Fo-POISSON



(e) Vary r, Go-ZIPFIAN



(f) Vary r, Fo-ZIPFIAN

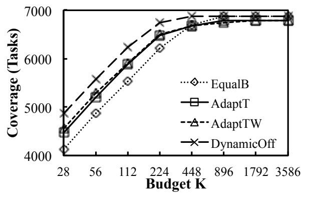
Figure 10: Performance of *AdaptT* in the dynamic-budget scenario.

Furthermore, to show the effectiveness of the adaptive algorithms in handling highly skew data, we evaluate *AdaptT* on Go-ZIPFIAN and Fo-ZIPFIAN. Figures 10e and 10f show the results by varying r. *AdaptT* obtains up to 36% and 40% improvements at $r = 1$, correspondingly.

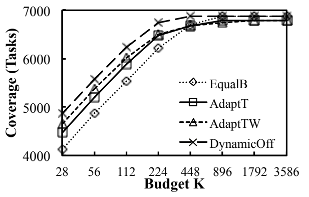
Table 3 shows the results on the iRain dataset. The results show small improvement of *AdaptT* over *EqualB* (i.e., up to 6%). This can be explained by the fact that each task can only be performed by less than two workers in average.

Historical Workload Improvement: We evaluate the performance of the workload strategy on real-world workload data. Figures 11a and 11b show the results by varying K on Go-POISSON and Go-CONST, respectively. *AdaptTW*, which uses historical optimal workload as the baseline budget strategy, marginally improves *AdaptT* on Go-POISSON. The reason is that POISSON distribution introduces randomness to the workload approach, which againsts the idea of leveraging the historical workload in *AdaptTW*. On the other hand, *AdaptTW* improves *AdaptT* by 7% on Go-CONST.

We conclude that *AdaptT* outperforms *EqualB*, closing the gap between the equal-budget baseline and optimal offline solutions. $AdaptTW$ further enhances $AdaptT$ by using optimal workload in the recent past.



(a) Vary K, Go-POISSON



(b) Vary K, Go-CONST

Figure 11: Performance of $AdaptTW$ on real-world data ($Q = 7$).

Runtime Measurements: Figure 12 shows the run times of our online algorithms by varying the number of tasks per time period. We observe that with the increase in the number of tasks, the runtime increases linearly. In addition, *EqualB* and *AdaptT* are shown to be very efficient (i.e., less than ten seconds), while the run time of $AdaptTW$ is much higher (i.e., over 100 seconds) due to the overhead in learning the optimal budget allocation in the recent past.

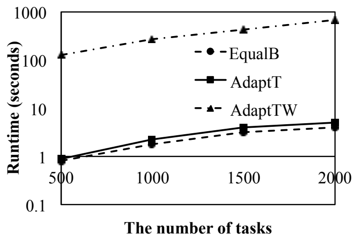


Figure 12: The average per-period run time on Go-CONST.

7. CONCLUSION AND FUTURE WORK

In this paper, we introduced a framework for crowdsourcing hyper-local information, where tasks can be performed by workers within their spatiotemporal vicinity and the number of assigned tasks can be maximized without exceeding the budget for activating workers. Two problem variants have been studied, one with a given budget at each time period, the other with a given budget for the entire campaign. We have shown that both variants are NP-hard in the offline case. In the online setting, several heuristics have been proposed, utilizing the spatial and temporal properties of tasks. Also, an adaptive strategy has been proposed to dynamically allocate budget over time. According to our extensive experiments, $AdaptTW$, which merits the dynamic budget strategy with temporal and workload heuristics, was shown to be the best technique.

As future work, we will extend the proposed framework to incorporate continuous utility functions where the utility of assignment decays depending on the distance from the worker to the task center. The intuition is that an assignment of a task to a nearby worker may yield higher utility than that of another worker farther from the task location. We will also consider non-uniform activation cost of the workers, which may represent the reputation or compensation demand of each worker. Finally, we will improve our heuristics by either linearly combining *Spatial* and *Temporal* or using multi-objective optimization.

8. REFERENCES

[1] 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.

[2] P. Cheng, X. Lian, Z. Chen, L. Chen, J. Han, and J. Zhao. Reliable diversity-based spatial crowdsourcing by moving workers. *arXiv preprint arXiv:1412.0223*, 2014.

[3] 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*. ACM, 2010.

[4] U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. Transdec: a spatiotemporal query processing framework for transportation systems. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 1197–1200. IEEE, 2010.

[5] B. Dorminey. Crowdsourcing for the weather. February 2014. [Accessed Feb. 2015].

[6] K. L. Elmore, Z. Flamig, V. Lakshmanan, B. Kaney, V. Farmer, H. D. Reeves, and L. P. Rothfusz. mping: Crowd-sourcing weather reports for research. *Bulletin of the American Meteorological Society*, 2014.

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

[8] D. S. Hochbaum. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In *Approximation algorithms for NP-hard problems*, 1996.

[9] M. D. Hoffman, E. Brochu, and N. de Freitas. Portfolio allocation for bayesian optimization. In *UAI*, pages 327–336. Citeseer, 2011.

[10] 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*. ACM, 2012.

[11] L. Kazemi, C. Shahabi, and L. Chen. GeoTruCrowd: trustworthy query answering with spatial crowdsourcing. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2013.

[12] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

[13] E. Malykhina. 8 apps that turn citizens into scientists. *Scientific American*, 2013.

[14] M. Musthag and D. Ganesan. Labor dynamics in a mobile micro-task market. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 641–650. ACM, 2013.

[15] W. H. Press. Bandit solutions provide unified ethical models for randomized clinical trials and comparative effectiveness research. *Proceedings of the National Academy of Sciences*, 106(52):22387–22392, 2009.

[16] H. Robbins. Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers*, pages 169–177. Springer, 1985.

[17] 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.

[18] H. To, L. Kazemi, and C. Shahabi. A server-assigned spatial crowdsourcing framework. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 2015.

[19] H. To, S. H. Kim, and C. Shahabi. Effectively crowdsourcing the acquisition and analysis of visual data for disaster response. *In proceeding of 2015 IEEE International Conference on Big Data*, 2015.

[20] J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In *Machine Learning: ECML 2005*, pages 437–448. Springer, 2005.