Real-Time Task Assignment in Hyperlocal Spatial Crowdsourcing under Budget Constraints

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

***Abstract*—Spatial Crowdsourcing (SC) is a novel platform that engages individuals in the act of collecting various types of spatial data. This method of data collection can significantly reduce cost and turnover time, and is particularly useful in environmental sensing, where traditional means fail to provide fine-grained field data. In this study, we introduce hyperlocal spatial crowdsourcing, where all workers who are located within the spatiotemporal vicinity of a task are eligible to perform the task, e.g., reporting the precipitation level at their area and time. In this setting, there is often a $budget$ constraint, either for every time period or for the entire campaign, on the number of workers to activate to perform tasks. The challenge is thus to maximize the number of assigned tasks under the budget constraint, despite the dynamic arrivals of workers and tasks as well as their co-location relationship. We study two problem variants in this paper: budget is constrained for every timestamp, i.e. $fixed$, and budget is constrained for the entire campaign, i.e. $dynamic$. For each variant, we study the complexity of its offline version and then propose several heuristics for the online version which 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 solutions.**

***Index Terms*—Crowdsourcing, Spatial Crowdsourcing, Mobile Crowdsensing, Online Task Assignment, Budget Constraints.**

I. Introduction

With the ubiquity of smart phones and the improvements of wireless network bandwidth, every person with a mobile phone can now act as a multimodal sensor collecting and sharing various types of high-fidelity spatiotemporal data instantaneously. In particular, crowdsourcing for weather information has become popular. With a few recent apps, such as mPING[1] and WeatherSignal[2], individual users can report weather conditions, air pollutions, noise levels, etc. In fact, Dorminey in [6] regards crowdsourcing as "the future of weather forecasting".

Through our collaboration with the Center for Hydrometeorology and Remote Sensing (CHRS)[3] at the University of California, Irvine, we have developed a mobile app, iRain[4], to perform *spatial crowdsourcing* for precipitation information. Unlike other weather crowdsourcing apps, iRain allows CHRS researchers to *request* rainfall information at specific locations and times where their global satellite precipitation estimation technologies[5] fail to provide real-time, fine-grained data. Individual iRain users around those locations can *respond* to those requests by reporting rainfall observations, e.g., heavy/medium/light/none, and they can also issue rainfall information requests by "subscribing" to regions of interest.

In general, spatial crowdsourcing (SC) [10] offers an effective data collection platform where data requesters can create spatial tasks dynamically and workers are assigned to tasks based on their locations. Figure 1 depicts the architecture of iRain. A requester issues a set of rainfall observation tasks to the SC-server (Step 1) where each task corresponds to a specific geographical extent, e.g., a circle. The workers 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 regions and sends the collected data back to the requester (Steps 2, 3).

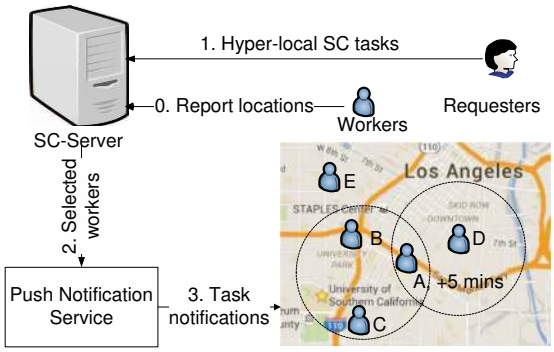


Fig. 1: Hyperlocal spatial crowdsourcing framework.

One major difference from existing SC paradigms [11], [10], [8], [17], [18], [20] is that workers in our paradigm do not need to travel to the exact task locations, e.g., to the centers of the circular regions, and are eligible to perform tasks as long as they are in close spatiotemporal vicinity of the tasks, e.g., enclosed in the circular regions[6]. We denote this new paradigm as *Hyperlocal Spatial Crowdsourcing*. The reason is twofold. Without requiring the workers travel physically, our paradigm lowers the threshold for worker participation and will potentially yield faster response. Furthermore, the requested data, e.g., rainfall or temperature, exhibits spatiotemporal continuity

[1] http://mping.nssl.noaa.gov/
[2] http://weathersignal.com
[3] http://chrs.web.uci.edu/
[4] https://play.google.com/store/apps/details?id=irain.app
[5] http://hydis.eng.uci.edu/gwadi/
[6] Tasks that require workers to physically travel to task locations, e.g., taking a picture of an event, are not considered in our problem setting.

978-1-4673-8779-8/16/$31.00 ©2016 IEEE

in measurement. Therefore, observations obtained at nearby locations, e.g., within certain distance to the task location, and close to the requested time, are sufficient to fulfill the task. For example, workers 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.

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 or 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 (shown in two circles).

The goal of our study 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 or during the entire campaign, i.e., under "budget" constraints. When tasks and workers are known *a priori*, 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 online 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. Previous heuristics in literature [18], [17], [10] do not consider the vicinity of tasks in space and time or the budget, thus cannot be applied to Hyperlocal SC.

The contributions of this paper are as follows: **1)** We provide a formal definition of Hyperlocal Spatial Crowdsourcing, where the goal is to maximize task coverage under budget constraints. We study two problem variants, i.e., given a budget constraint for each time period or for the entire campaign. We show both variants are NP-hard in the offline scenario. **2)** When a budget is specified for each time period, we propose three heuristics for the online setting, i.e., *Basic*, *Temporal*, and *Spatial*. The temporal heuristic favors the tasks which will soon expire, while the spatial heuristic favors the tasks that may not co-locate with future workers. **3)** When a budget is specified for the entire campaign, we devise an adaptive strategy based on the contextual bandit to dynamically allocate the total budget to a number of time periods. Our strategy strikes a balance between *exploitation* and *exploration* and captures 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 approaches.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III provides notations for Hyperlocal SC problem. In Section IV and V, we study two problem variants and their online solutions. We report our experimental results in Section VI, provide discussion in Section VII, and conclude the paper in Section VIII.

II. Related work

There have been extensive studies regarding task assignment in generic crowdsourcing. However, only recently spatial crowdsourcing (SC) has gained popularity in both research community (e.g., [21], [17], [10]) and industry (e.g., TaskRabbit, Gigwalk [13]). A recent survey in [18] distinguishes SC from related fields, including generic crowdsourcing, participatory sensing, volunteered geographic information, and online matching. Research efforts on SC have focused on different aspects, such as scalable task assignment [1], [10], task scheduling [5], trust [11], and privacy [17]. In [10], Kazemi and Shahabi proposed task assignment problem whose goal is to maximize the number of assigned tasks, and Alfarrarjeh et al. [1] scaled out the assignment algorithm in a distributed setting. The trust issues in SC have been studied in [11], where one solution is having tasks performed redundantly by multiple workers. Recently in [3], Cheng et. al. study reliable task assignment in SC is to maximize both the confidence of task completion and the diversity quality of the tasks. However, the trust and reliability of workers is beyond the scope of our work; if there are multiple reports for one task, the SC-server will simply send all available reports to the task requester.

Several works [14], [21] studied the problem of selecting workers with budget constraints. However, those studies focus on offline participant selection problem while our focus is to propose online solutions. Furthermore, the problem settings in those studies differ from ours in several aspects. Sensing tasks in [14] are represented by non-overlapping regions while tasks in our study can overlap spatially thus more challenging for optimization. Zhang et. al. [21] studied the problem of selecting a minimum number of workers to minimize the overall incentive payment while satisfying a probabilistic coverage requirement; however, in our problem, the number of workers to be selected is constrained by a predefined budget.

Our work is also related to the problem of matching workers with tasks [8], [20]. In particular, He et. al. [8] studied the problem of task allocation that maximizes the reward of the SC platform given a time constraint for each worker. Xiao et. al. [20] proposed a task assignment problem that minimizes the average makespan of all assigned tasks. Unlike those studies, SC workers in our setting need not to travel to task locations. Furthermore, our aim is different from the aforementioned studies, which is to maximize task coverage.

III. Preliminaries

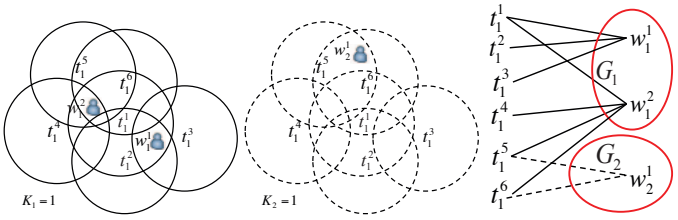
We first introduce concepts and notations used in this paper. A **task** is a query of certain hyperlocal information, e.g., precipitation level at a particular location and time. For simplicity, we assume that the result of a task is in the form of a numerical value, e.g., *0=rain,1=snow,2=none*[7]. Specifically,

[7]Remote sensing techniques based on satellite images cannot differentiate between rain and snow.

every task comes with a pre-defined region where any enclosed worker 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. 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



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

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

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 eligible to perform t_1^1, t_1^2 and t_1^3 while w_1^2 is qualified for 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, 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 un-expired 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 I: The coverage instance set of the example in Figure 2.

The coverage instance sets for the example in Figure 2 are illustrated in Table I. 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.

Again, 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 selected workers (Step 3 in Figure 1), or the *rewards* paid to the workers.

IV. Fixed Budget

The first variant of the maximum task coverage problem is when a budget constraint is given for each time period. We first study the problem complexity for the offline scenario and then propose heuristics for the online scenario.

A. 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 (f*MTC*) problem is to select a set of workers L_i at every 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 $|L_i| \leq K_i$.

We prove in our technical report [16] that the $fMTC$ problem is NP-hard by a reduction from the *maximum coverage with group budgets* constraints problem (MCG) [2]. The greedy algorithm is shown in [2] to provide 0.5-approximation for MCG. For example, the greedy solution in Figure 2c is $\{w_1^1, w_2^1\}$. However, the approximation ratio only holds in the offline scenario where the server knows $apriori$ the coverage instance set for every time period.

B. Online Scenario

In the online scenario where workers and tasks arrive dynamically, it becomes more challenging to achieve the global optimal solution for Problem 1. Since the server does not have prior knowledge about future workers and tasks, it tries to optimize task assignment locally at every time period. However, the optimization within every time period, similar to the *maximum coverage problem* (MCP) [7], is also NP-hard. A greedy algorithm [7] was proposed to achieve an approximation ratio of 0.63, by choosing a set which contains the largest number of *uncovered* elements at each stage. The results in [7] showed that the greedy algorithm is the best-possible polynomial time approximation algorithm for MCP.

Below we propose several greedy heuristics to solve the online *fMTC* problem, namely *Basic*, *Spatial* and *Temporal*.

1) Basic Heuristic: The *Basic* heuristic solves the online *fMTC* problem by using the greedy algorithm [9] for every time period. At each stage, *Basic* selects the worker that covers the maximum number of *uncovered* tasks, depicted in Line 10 of Algorithm 1. For instance, in Figure 2a, w_1^2 is selected at the first stage. At the beginning of each time period, Line 4 removes expired tasks from the previous time period. Line 5 adds unassigned, unexpired tasks to current task set. Line 12 outputs the covered tasks C_i per time period which will be used as the main performance metric in Section VI. The algorithm terminates when either running out of budget or all the tasks are covered (Line 9).

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 $R = T'_i$
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: $R \leftarrow R - C(w_i^j)$; $L_i \leftarrow L_i \cup \{w_i^j\}$
12: $C_i \leftarrow \bigcup_{w_i^j \in L_i} C(w_i^j)$
13: Keep uncovered tasks $U_i \leftarrow T'_i - C_i$

Basic can achieve fast task assignment by simply counting the number of tasks covered by each worker (Line 10). However, it treats all tasks equally without considering the spatial and temporal information of each task, i.e., location and deadline. For example, a task located in an "worker-sparse" area may not be assigned in the future due to lack of nearby workers and thus should be assigned with higher priority at the current iteration. Similarly, tasks that are expiring soon should be assigned with higher priorities. Consequently, the priority of a worker is high if he covers a larger number of high priority tasks. Below we introduce two assignment heuristics that explicitly model the task priority given its spatial and temporal characteristics .

2) Temporal Heuristic: One approach to prioritizing tasks is by considering their temporal urgency. The intuition is that a task which is further away from its deadline is more likely to be covered in the future, and vice versa. As a result, near-deadline tasks should have higher priorities to be assigned than others. Consequently, a worker who covers a large number of soon-to-expire tasks should be preferred for selection. Based on the above intuition, we model the priority of a worker w_i^j based on the remaining time of each task his covers as follows

$$priority(w_i^j) = \sum_{t_i^k \in C(w_i^j) \cap R} \frac{1}{t_i^k.(s+\delta) - i} \tag{3}$$

The *Temporal* heuristic adapts *Basic* by selecting the worker with maximum *priority* at each stage. For instance, given two workers w_1^1 and w_1^2 at time s_1, where $C(w_1^1) = \{t_1^1, t_1^2\}$ and $C(w_1^1) = \{t_1^3\}$. Suppose both t_1^1 and t_1^2 expire in 5 time periods and t_1^3 expires in 2 time periods. The *Temporal* heuristic chooses w_1^2 over w_1^1 as their priorities are 0.5 and 0.4, respectively. To implement *Temporal*, Line 10 in Algorithm 1 can be updated to select the worker with maximum priority defined as in Equation 3. We will empirically evaluate all heuristics in Section VI.

3) Spatial Heuristic: To maximize task assignment in the long term, we also consider the "popularity" of a task location as an indicator of whether the task can be assigned to future workers. Accordingly, we can spend the budget for the current time period to assign those tasks which can be only covered by existing workers. The "popularity" of a task region can be measured using Location Entropy [4], which captures the diversity of visits to that region. A region has a high entropy if many workers visit with equal probabilities. In contrast, a region has a low entropy if there are only a few workers visiting. We define the *region entropy* of an given task as follows.

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

$$RE(t) = -\sum_{w \in W_t} P_t(w) \times logP_t(w) \tag{4}$$

For efficient evaluation, $RE(t)$ can be approximated by aggregating the entropies of 2D grid cells within the task region $R(t.l, r)$ and the cell entropies can be precomputed using historical data. Since any worker located inside $R(t.l, r)$ can perform task t, t is likely to be covered in the future as long as one grid cell inside $R(t.l, r)$ is "popular" among workers. An illustrative example of the computation of the region entropy of a task using pre-computed cell entropies can be found in our technical report [16]. With the region entropy of every task covered by worker w_i^j, his priority can be calculated as follows

$$priority(w_i^j) = \sum_{t_i^k \in C(w_i^j) \cap R} \frac{1}{1 + RE(t_i^k)} \tag{5}$$

Note that the constant 1 is needed to avoid division by zero. Consequently, the *Spatial* heuristic greedily selects the worker with maximum *priority* at each stage. Line 10 in Algorithm 1 can be modified to reflect the spatial priority of each worker.

V. DYNAMIC BUDGET

The second problem variant we study is more general, where a budget constraint is given for the entire campaign. This relaxation often results in higher task coverage. For example, in Figure 2, if budget 1 is given at every time period, we select w_1^1 and w_2^1 and obtain the coverage of 5. However, the dynamic-budget variant yields higher coverage of 6 by selecting w_1^1 and w_1^2 at time 1. Below we study the problem

complexity in the offline scenario and propose adaptive budget allocation strategies for the online scenario.

A. Offline Scenario

Problem 2 (Dynamic-budget Maximum Task Coverage): The dynamic-budget maximum task coverage problem (*dMTC*), is similar to *fMTC*, except the total budget K is specified for the entire campaign, i.e., $\sum_{i=1}^{Q} |L_i| \leq K$.
In the offline scenario where the server is clairvoyant about the future workers and tasks, we prove the *dMTC* problem is NP-hard in our technical report [16]. This can be shown by a reduction from the maximum coverage problem (MCP) [7].

B. Online Scenario

The challenge of the online *dMTC* problem is to allocate the overall budget K over Q time periods ($K \geq Q$) optimally, despite the dynamic arrivals of workers and tasks. Below we introduce several budget allocation strategies. Once a budget is allocated to a particular time period, we can adopt previously proposed heuristics, i.e., *Basic, Spatial, Temporal*, to select the best worker.

The simplest strategy, namely *Equal*, equally divides K to Q time periods; each time period 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 allocate a budget to each time period proportional to the number of available tasks at that time period, i.e., $\frac{|T_i|}{|T|}K$, where $|T|$ is the total number of tasks. However, $|T|$ is not known *a priori*. Furthermore, we may still over-allocate budget to any time period with large $|T_i|$, if none of the tasks can be covered by any workers (or all the tasks can be covered by 1 worker). We cannot allocate budget optimally without looking at the coverage instance set at each time period.

1) Adaptive Budget Allocation: To maximize task assignment, we need to adaptively allocate the overall budget and consider the "return" of selecting every worker, i.e., the worker priority, given the dynamic coverage instance set at every time period. 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 $\{K^{base}[t], t = 1, \ldots, Q\}$, e.g., the *Equal* strategy . Given a certain baseline $\{K^{base}[t]\}$, δ_K is the difference between the cumulative baseline budget and the actual budget spent up to time period s_i. Formally, at any time period s_i,

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

A positive δ_K indicates budget is under-utilized, and vice versa. Another notion is **delta gain**, denoted as δ_λ, which represents the return of a worker currently being considered (λ_l) compared to the ones selected in the past ($\overline{\lambda_{l-1}}$). Formally,

$$\delta_\lambda = \lambda_l - \overline{\lambda_{l-1}} \tag{7}$$

where λ_l is the gain of the current worker, calculated by any previously proposed local heuristic, i.e., as $|priority(w_i^j)|$. $\overline{\lambda_{l-1}}$ is the average gain of previously added workers, i.e., $\overline{\lambda_{l-1}} = \frac{1}{l-1}\sum_{t=1}^{l-1}\lambda_t$. A positive δ_λ indicates the current worker has higher priority than the historical average, and vice versa.

Based on the contextual information δ_K and δ_λ at each stage of worker selection, we examine all available workers at the currently time period and decide whether to allocate budget 1 to selecting any worker. Intuitively, when both δ_K and δ_λ are positive, i.e., the budget is under-utilized and a worker has higher priority, the selection of the considered worker is favored. When both are negative, it may not be worthwhile to spend the budget. The other cases when one is positive and the other is negative are more complex, as we would like to spend budget on workers with higher priority but also need to save budget for future time periods in case better worker candidates arrive.

Our solution to the sequential decision problem is inspired by the well-know multi-armed bandit problem (MAB), which has been widely studied and applied to decisions in clinical trials, online news recommendation, and portfolio design. ϵ-greedy, which achieves a trade-off between exploitation and exploration, proves to be often hard to beat by other MAB algorithms [19]. Hence, we propose an adaptive budget allocation strategy, based on *contextual ϵ-greedy* algorithm [12]. We illustrate our solution in Figure 3.

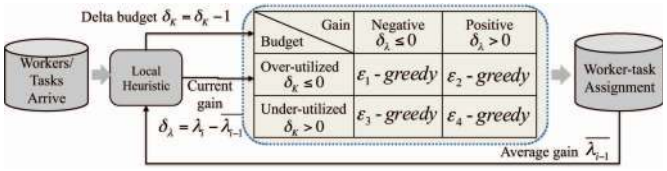


Fig. 3: Adaptive budget allocation based on contextual ϵ-greedy.

At each stage of the local heuristic, a binary decision to make is whether to allocate budget 1 to activate the current worker with the highest priority. The contextual ϵ-greedy algorithm allows us to specify an exploration-exploitation ratio, i.e., ϵ, based on the worker's context, i.e., δ_K and δ_λ. As depicted in Figure 3, an ϵ_i-greedy algorithm is used to determine whether to select the current worker based on his δ_K and δ_λ. For each case, a YES decision is made with $1-\epsilon_i$ probability and a NO decision with ϵ_i probability. By default we set $\epsilon_1 = 1$ and $\epsilon_4 = 0$ to reflect NO and YES decisions, respectively, as discussed before. When δ_K and δ_λ have different signs, the decision is not as straightforward as the other cases and thus we set $\epsilon_2 = \epsilon_3 = 0.5$ to allow YES and NO decisions with equal probabilities. The pseudocode of our adaptive algorithm can be found in our technical report [16].

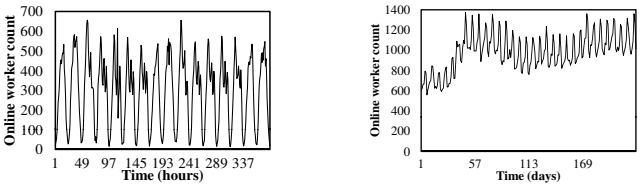
2) Historical Workload: Previously our solution is simplified by considering $\{K^{equal}[t]\}$ as the baseline budget strategy. Since human activity exhibits temporal patterns, understanding those patterns may help guide budget allocation. Therefore we propose to compute a baseline budget strategy with historical data that captures the expected worker/task patterns.

Musthag et al. [13] show the time-of-day usage patterns of workers in mobile crowdsourcing applications. The activity

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 II: Summaries of real-world datasets.

peaks are between 4 to 7 pm when workers leave their day jobs. Similar patterns are observed in Foursquare and Gowalla data sets in Figure 4. Figure 4a shows the hourly count of check-ins present three peaks, i.e., during lunch and morning/afternoon commute. In Figure 4b, we can observe peak check-in activities during weekends.



(a) Foursquare, 16x24 hours (b) Gowalla, 32x7 days

Fig. 4: Daily and weekly human activity patterns.

With historical worker and task information, we can leverage the optimal budget allocation strategy in the recent past and use it as the baseline strategy in Equation 6. We propose to learn the budget allocation of previous time periods, namely *workload*, using the greedy algorithm for the offline *dMTC* problem. To guide future budget allocation decisions, the previous workload $K^{prev}[]$ will be used as the baseline in Equation 6. We will empirically evaluate our proposed solutions in the next section.

VI. Performance Evaluation

A. Experimental Methodology

We adopted real-world datasets from location-based applications, summarized in Table II, which have been used in [18], [17], [10] to emulate spatial crowdsourcing (SC) workers and tasks [15]. We consider Gowalla (or Foursquare) users as SC workers and the venues as tasks. The Gowalla dataset contained check-ins for 224 days in 2010, including more than 100,000 spots (e.g., restaurants), within the state of California. By considering each day as a unit time period, all the users who checked in during a day are available workers for that time period in our setting. Foursquare dataset contains the check-in history of 45,138 users to 89,968 venues over 384 hours in Pittsburgh, Pennsylvania. We considered each hour as a unit time period for this dataset.

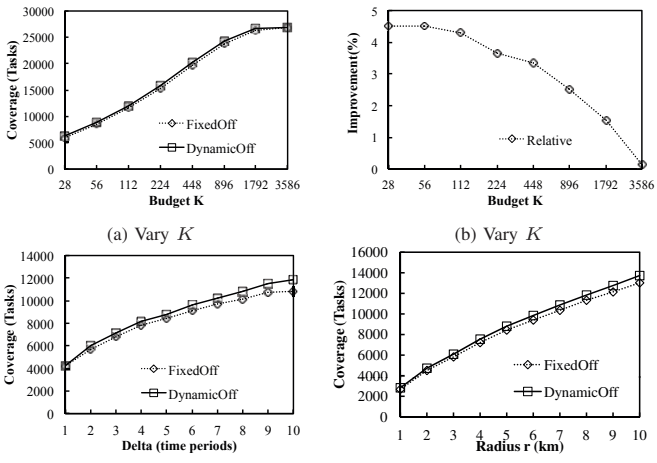
We generated a range of datasets by utilizing real-world worker/task spatial distributions and varying their arrival rate. For arrival rate, we only needed to generate task count per time period as the worker counts can be obtained from Gowalla (mean=991) and Foursquare (mean=291). CONST, POISSON (default), ZIPFIAN and COSINE distributions with mean=1000 were adopted to generate the number of available tasks for every time period. The distributional parameters and their figures can be found in our technical report [16]. For example, Go-POISSON uses Gowalla for the spatial distributions and the worker arrival rate (Fig. 4b) and POISSON for the task arrival rate.

In all of our experiments, we varied the total 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[9]. 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.

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

B. Offline Solutions

We compared the offline solutions to the two problem variants, *fMTC* (Section IV-A) and *dMTC* (Section V-A), using the greedy algorithm. Figure 5a 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 5b. This effect can be explained by the diminishing return property. That is, the coverage differences between the algorithms are small with high budgets.



(a) Vary K (b) Vary K

(c) Vary δ (d) Vary r

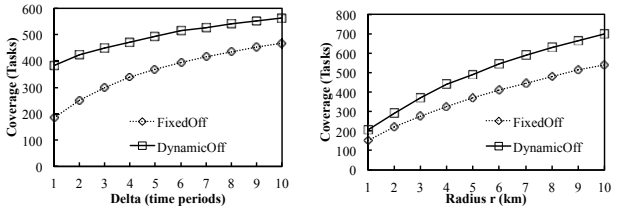
Fig. 5: Performance of offline solutions with Go-POISSON.

We also evaluated the offline solutions by varying task duration δ. As expected, Figure 5c 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 5d, when r increases, every task can be covered by more workers, which yield higher coverage.

[8] MTD: Mean Travel Distance [17]

[9] The choices of r and δ values are defined by the CHRS experts.

We do 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 6). Unlike the result in Figure 5c, Figure 6a 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 allocate more budget to those time periods to cover more tasks.

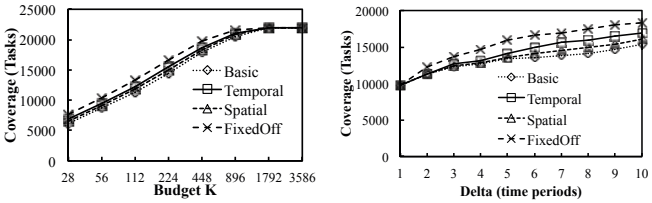


(a) Vary δ (b) Vary r

Fig. 6: Performance of offline solutions with Go-ZIPFIAN.

C. Online Solutions

The Performance of Heuristics: We evaluated the performance of the online heuristics from Section IV, *Basic*, *Spatial* and *Temporal*. Figures 7a 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 7b 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 similarly 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.

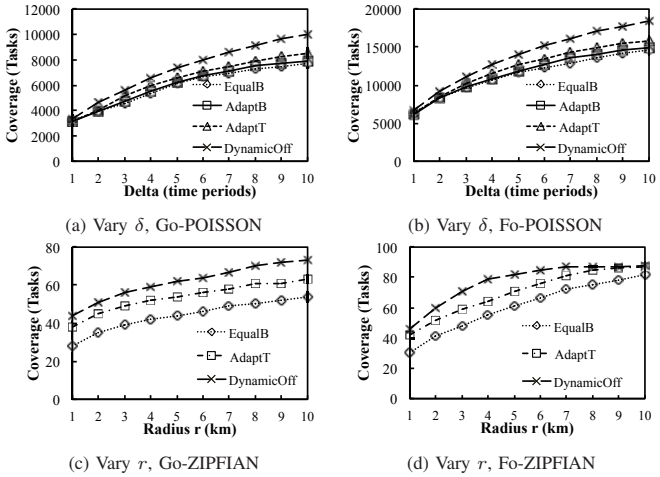


(a) Vary K, Go-POISSON (b) Vary δ, Go-POISSON

Fig. 7: Performance of heuristics in the fixed-budget scenario.

Adaptive Budget Strategy: We evaluated the performance of the adaptive algorithms in Section V-B1. *EqualB* refers to the algorithm that divides the budget equally to time periods and runs *Basic* local heuristic, whereas *AdaptB* and *AdaptT* adopt adaptive budget allocation with *Basic* and *Temporal*, respectively. Figures 8a and 8b 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* outperforms *AdaptB*, we hereafter show only the results of *AdaptT*.

Furthermore, to show the effectiveness of the adaptive algorithms in handling highly skew data, we evaluated *AdaptT*



(a) Vary δ, Go-POISSON (b) Vary δ, Fo-POISSON

(c) Vary r, Go-ZIPFIAN (d) Vary r, Fo-ZIPFIAN

Fig. 8: Performance of *AdaptT* in the dynamic-budget scenario.

on Go-ZIPFIAN and Fo-ZIPFIAN. Figures 8c and 8d show the results by varying r. *AdaptT* obtains up to 36% and 40% improvements at $r = 1$, correspondingly.

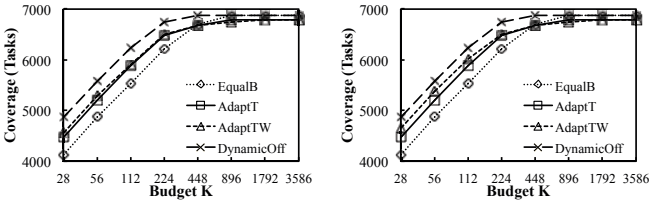
We also evaluated our algorithms on the iRain dataset, which includes 1,355 workers and 385 tasks. Since the original dataset is small, we generated a synthetic dataset with similar spatial distributions of workers and tasks. We discretized the entire space into 200x200 grid. For each time instance, we randomly generated workers and tasks within the grid cells proportional to the dataset density. The worker/task locations are randomly distributed within each cell. Table III summarizes the results. We observe a small improvement of *AdaptT* over *EqualB*, e.g., by 6%. The reason is that each task can only be performed by less than two workers on average.

r	EqualB	AdaptT	δ	EqualB	AdaptT
1	8932	9396	1	18564	19251
5	24819	25321	5	24620	25112
10	26859	27442	10	24819	25274

TABLE III: Task coverage of *AdaptT* with iRain-POISSON, K=56, Q=28.

Historical Workload Improvement: We evaluated the performance of the workload strategy on real-world workload data. Figures 9a and 9b 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 much randomness to the workload, which makes it challenging to benefit from historical data in *AdaptTW*. On the other hand, *AdaptTW* improves *AdaptT* by 7% with Go-CONST.

Runtime Measurements: Figure 10 shows the run time performance 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.



(a) Vary K, Go-POISSON (b) Vary K, Go-CONST

Fig. 9: Performance of *AdaptTW* with real-world data ($Q = 7$).

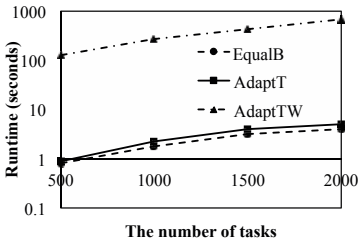


Fig. 10: Average run time per time period with Go-CONST.

VII. Discussion

Unlike Mobile Crowdsourcing, our model does not consider *individual* worker mobility, i.e. the worker's trajectory. The reason is that the workers in our setting do not need to travel to the locations of tasks and thus minimizing individual travel distance is not our objective. Furthermore, a worker's trajectory within the task region, or the intersection of several task regions, does not affect his eligibility to perform the task(s). The effect takes place only when he leaves or joins the region. In fact, our *Spatial* heuristic (Section IV) considers worker *population* mobility by prioritizing tasks whose locations are not likely to be visited by many workers in the future.

As for future work, we will study the following extensions. First, we plan to incorporate continuous utility functions where the utility of a task assignment depends on the distance between the worker and the task. The intuition is that a task assigned to a nearby worker may yield higher utility than assigned to another worker farther from the task location. Second, we will consider non-uniform activation cost of the workers, which represents the reputation or the compensation demand of each worker. Our local heuristics and adaptive budget strategy should be adjusted to reflect the weight of each worker. Finally, we will formulate a multi-objective optimization problem to avoid repetitive activations of the same workers. We will minimize worker overload, which may result in either low quality responses or rejected tasks since the worker may feel annoyed or stressed by repetitive requests.

VIII. Conclusion

We introduced hyperlocal spatial crowdsourcing, 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. We studied two problem variants, i.e., *fMTC* with a given budget for each time period and *dMTC* with a given budget for the entire campaign. We showed that both variants are NP-hard to solve offline and proposed several local heuristics and dynamic budget allocation for the online scenario which utilize the spatial and temporal properties of workers/tasks. We conducted extensive experiments and concluded that *AdaptTW*, which merits the temporal local heuristic and dynamic budget allocation with workload baseline, is the superior technique.

IX. Acknolwdgement

This research has been funded by NSF grants IIS-1320149 and CNS-1461963, the USC Integrated Media Systems Center (IMSC), and unrestricted cash gifts from Google, Northrop Grumman, Microsoft, and Oracle. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any of the sponsors such as NSF.

References

[1] A. Alfarrarjeh, T. Emrich, and C. Shahabi. Scalable spatial crowdsourcing: A study of distributed algorithms. In *Mobile Data Management (MDM), 2015 16th IEEE International Conference on*, volume 1, pages 134–144. IEEE, 2015.

[2] 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.

[3] 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.

[4] 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.

[5] D. Deng, C. Shahabi, and U. Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *The 21st ACM SIGSPATIAL GIS 2013*, pages 314–323. ACM, 2013.

[6] B. Dorminey. Crowdsourcing the weather. February 2014. http://www.forbes.com/sites/brucedorminey/2014/02/26/crowdsourcing-as-the-future-of-weather-forecasting/ [Accessed Jan. 2016].

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

[8] S. He, D.-H. Shin, J. Zhang, and J. Chen. Toward optimal allocation of location dependent tasks in crowdsensing. In *INFOCOM, 2014 Proceedings IEEE*, pages 745–753. IEEE, 2014.

[9] 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.

[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*, 2012.

[11] L. Kazemi, C. Shahabi, and L. Chen. GeoTruCrowd: trustworthy query answering with spatial crowdsourcing. In *The 21st ACM SIGSPATIAL GIS 2013*, 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] 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.

[14] Z. Song, C. H. Liu, J. Wu, J. Ma, and W. Wang. Qoi-aware multitask-oriented dynamic participant selection with budget constraints. *Vehicular Technology, IEEE Transactions on*, 63(9):4618–4632, 2014.

[15] H. To, M. Asghari, D. Deng, and C. Shahabi. SCAWG: A toolbox for generating synthetic workload for spatial crowdsourcing. In *CROWDBENCH 2016*. IEEE.

[16] H. To, L. Fan, L. Tran, and C. Shahabi. Real-time task assignment in hyper-local spatial crowdsourcing under budget constraints. *Technical Report ID 15-962, University of Southern California*, 2015. http://www.cs.usc.edu/assets/007/97707.pdf.

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

[20] M. Xiao, J. Wu, L. Huang, Y. Wang, and C. Liu. Multi-task assignment for crowdsensing in mobile social networks. 2015.

[21] D. Zhang, H. Xiong, L. Wang, and G. Chen. CrowdRecruiter: selecting participants for piggyback crowdsensing under probabilistic coverage constraint. In *ACM UbiCom 2016*, pages 703–714. ACM, 2014.