Minimizing Maximum Delay of Task Assignment in Spatial Crowdsourcing

Zhao Chen
HKUST
Hong Kong, China
zchenah@cse.ust.hk

Peng Cheng
HKUST
Hong Kong, China
pchengaa@cse.ust.hk

Yuxiang Zeng
HKUST
Hong Kong, China
yzengal@cse.ust.hk

Lei Chen
HKUST
Hong Kong, China
leichen@cse.ust.hk

***Abstract*—Spatial crowdsourcing services, such as Uber and Grabhub, become popular recently. Task assignment plays an important role in offering high-quality services. However, most of the existing solutions for task assignment only focus on the entire performance of the platform and do not optimize the maximum assignment delay. As a result, they cannot handle some real world scenarios which require minimizing the maximum delay in task assignment. In this paper, we study the minimizing maximum delay spatial crowdsourcing (MMD-SC) problem and propose solutions aiming at achieving a worst case controlled task assignment.**

The MMD-SC problem assumes that both workers and requesters come dynamically and considers not only the workers' travel costs but also the buffering time of tasks, thus it is very challenging due to two-sided online setting. To address these challenges, in this work, we propose a space embedding based online random algorithm with a competitive ratio of $O(\log n)$ and two efficient heuristic algorithms, namely *the threshold based greedy approach* and *the batch-based approach*. In addition, we demonstrate the effectiveness and efficiency of our methods via extensive experiments on both synthetic and real datasets.

I. Introduction

Following the blossom of mobile Internet and the popularity of sharing economy, various applications (e.g., Uber [7] and TaskRabbit [5]) allow users to participate in some location-based tasks close to their current positions, such as hailing cars (e.g., DiDi Chuxing [1]), delivering food (e.g., Ele.me [2]), and reporting the daily menu of a restaurant (e.g., Gigwalk [3]). Recently, a novel framework, namely *spatial crowdsourcing*, is proposed to study the problem of dispatching spatial tasks having attracted much attention from academia and industry. In spatial crowdsourcing platforms, there are typically three parties: crowd workers (e.g., Uber drivers), task requesters (e.g., passengers), and the platform (e.g., Uber server). Both workers and requesters reveal their locations to the platform who then assigns tasks to workers with different goals (e.g., maximizing the total assigned tasks) subject to spatial/temporal constraints (e.g., deadlines of tasks).

Existing work mostly focused on optimizing the entire performance of the whole platform, such as the total travel cost [40], [42], the overall utility/quality score [14], [33] and the total number of assigned tasks [24], [36], [41]. However, in some scenarios, the worst individual performance is critical. For example, one key issue affecting the user experience of online car-hailing system is the *request delay time* (i.e. how long does a rider wait from the request is received by the platform till he/she is picked up by an assigned car). Riders usually cannot bear the unusual long waiting time, which causes people crazy and permanently leave the platform. In addition, the maximum request delay time is also a critical metric in online food delivery platforms. Usually the longest delivery time is promised lower than a given value to guarantee the food remaining fresh. To alleviate the worst individual waiting time problem, in this paper, we consider an important problem in spatial crowdsourcing systems, namely *minimizing maximum delay spatial crowdsourcing* (MMD-SC), which assigns workers to tasks with a goal to minimize the maximum delay. To further illustrate this motivation, we go through the following example.

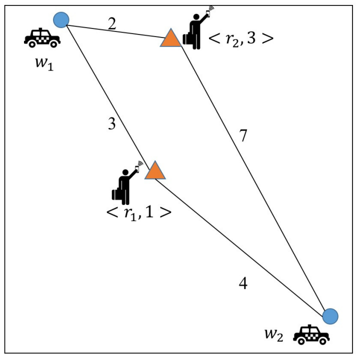


Fig. 1. An example of the MMD-SC problem.

Example 1. *(Minimizing Maximum Delay Spatial Crowdsourcing Example) In this example of minimizing maximum delay spatial crowdsourcing problem, there are two riders r_1 and r_2 (marked with yellow triangles), and two taxis, w_1 and w_2 (marked with blue circles) as shown in Figure 1. Specifically, for each rider, the tuple near to him/her denoted as $\langle$rider ID, request timestamp$\rangle$ representing that a rider with the given rider ID sends a request at the request timestamp. For example, rider r_1 requests a taxi on timestamp 1. In addition, the numbers near to the links indicate the travel time from two linked taxi and rider.*

The riders send their requests to the car-hailing system server. Then, the server tries to assign a suitable taxi to the rider to deliver him/her to his/her destination. Usually, the time period from the request being posted to the server to the assigned taxi picking up the rider is considered as the delay of *pickup, which affects the user experience for the car-hailing system dramatically. Thus, the maximum delay needs to be as small as possible.*

In the car-hailing system, riders post their requests at different timestamps and the server needs to assign taxis to suitable riders at different timestamps. One simple greedy strategy may assign each rider with the closest taxi to him/her once his/her request is received. In this example, this strategy can achieve a very bad result of assigning w_1 to r_1 and w_2 to r_2 having the maximum delay of 7. The optimal solution is to assign w_2 to r_1 and w_1 to r_2, which just has the maximum delay of 4.

In this paper, we study the minimizing the maximum delay spatial crowdsourcing (MMD-SC) problem, which matches the available workers to the requesters such that the maximum delay is minimized. The delay consists of two parts: 1) time since a request comes till he/she is assigned; 2) the assigned worker's traveling time (i.e., the travel time of a taxi from its current location to the position of the assigned rider).

The goal of MMD-SC is similar to *the bottle matching problem* [13], but the *online* and *waiting time* settings differentiate MMD-SC from existing problems. There are two major challenges need to be addressed in the MMD-SC problem: *(i) How to effectively bound the performance under the two sided online setting? (ii) How to consider the assign delay time and traveling time together so that the total delay can be reduced?* To solve the MMD-SC problem, we propose two efficient heuristic algorithms and a Hierarchically well-Separated Tree (HST, [19]) based online random algorithm with a competitive ratio of $O(\log n)$. We summarize the main contributions of this work as follows.

- We formally define the MMD-SC problem which is the first work focusing on optimizing the worst task performance (the maximum delay) for task assignment in spatial crowdsourcing in Section II.
- We show the deterministic bound of the MMD-SC problem in Section II-C.
- We propose two basic algorithms in Section III. In addition, we propose a space embedding based online random algorithm that can achieve results with at most $O(\log n)$ expected times of delay comparing to the optimal results in offline solutions in Section IV-C.
- We verify the effectiveness and efficiency of our methods on both real datasets and synthetic datasets in SectionV.

In addition, Section VI reviews the related studies and Section VII concludes this paper.

II. Problem Settings

We first formally define the MMD-SC problem, and then discuss how to measure the performance of an online (random) task assignment algorithm through a competitive ratio. Finally, we show the performance bound for deterministic algorithms is unacceptable for practical usage.

A. Definitions

We first introduce three basic concepts: *working space*, *crowd worker* and *task requester*.

Definition 1. *(Working Space) A working space is a metric space $S = \langle L, d\rangle$ where $L = \{l_1, l_2, ..., l_n\}$ is the n points inside the space and d is the distance function with $d(l_i, l_j)$ denote the travel cost between l_i and l_j.*

A working space defines the area where workers and tasks locate in and the travel cost between any two locations. It can be any kind of metric space, such as a 2D Euclidean space or a road network. In this paper, we consider the travel cost as the travel time between any two locations. In addition, we assume the travel velocities of workers are known, then we can easily convert the travel time to the travel distances. Without loss of generality, in the rest of paper, we also use $d(l_i, l_j)$ to denote the travel time from location l_i to location l_j.

Definition 2. *(Spatial Worker) A spatial worker, denoted by w_i, locates at location l_i and appears on the platform at timestamp t_i.*

In spatial crowdsourcing systems, workers are free to join and leave. The system can only assign tasks to a worker after he/she appears in the system. A worker w_i can move to any other location l_x within time $d(l_i, l_x)$.

Definition 3. *(Spatial Task/Request) A spatial crowdsourcing task or a request, denoted by r_j, appears on the platform at timestamp t_j and specifies a working location l_j.*

Definition 4. *(Assignment Triple) An assignment triple $a_{ijx} = \langle w_i, r_j, t_x\rangle$ indicates that the platform assigns a worker w_i to a task r_j on time t_x.*

Worker w_i needs to move to the location l_j of task r_j immediately after receiving the assigned task at timestamp t_x. We note the cost/delay of $a_{ijx} = \langle w_i, r_j, t_x\rangle$ as follows:

$$cost(a_{ijx}) = t_x - t_j + d(l_i, l_j), \tag{1}$$

which indicates the total time that r_j waits before worker w_i arrive at location l_j, i.e. the server processing time from t_i to t_x, and the travel time $d(l_i, l_j)$ of worker w_i. Note that, a valid assignment triple a_{ijx} requires that the assignment is placed after worker w_i and request r_j appear in the platform (i.e., $t_x \geq t_i$ and $t_x \geq t_j$).

Definition 5. *(The Minimizing maximum delay spatial crowdsourcing (MMD-SC) problem) In a working space S, given a set of k workers W and a set of k tasks R, the MMD-SC problem is to find a set A of k assignment triples such that the maximum delay $\max_{a_{ijx}\in A} cost(a_{ijx})$ is minimized.*

Note that, because each task needs one worker and each worker can conduct one task, k assignment triples can cover k workers and k tasks.

B. Performance Measurement

The performance of online algorithms is usually measured in terms of their competitive ratios [12]. Following the convention of recent online matching works [18] and [10], we formally define the competitive ratio as below:

Definition 6. *(Competitive Ratio) Let $\mathcal{A}$ be an (random) online algorithm and $\mathcal{A}^*$ be the optimal offline algorithm. We say $\mathcal{A}$ is α-competitive if for any* MMD-SC *problem instance $\mathcal{I}$, even generated by an oblivious adversary who knows $\mathcal{A}$ but not its random choices, the result achieved by $\mathcal{A}$ has the Competitive Ratio (CR) of α as follows:*

$$\alpha \geq \max_{\forall \mathcal{I}} \frac{cost_{\mathcal{A}}(\mathcal{I})}{cost_{\mathcal{A}^*}(\mathcal{I})}, \tag{2}$$

where $cost_{\mathcal{A}}(\mathcal{I})$ and $cost_{\mathcal{A}^*}(\mathcal{I})$ indicate the (expected) maximum delays achieved by $\mathcal{A}$ and $\mathcal{A}^*$, respectively.

Note that, in Definition 6 $cost_{\mathcal{A}}(\mathcal{I})$ is the expected maximum delay achieved by $\mathcal{A}$, when it is an online random algorithm.

C. Discussion on the Lower Bound of Competitive Ratio

In [25], the authors proved that the lower bound of the competitive ratio achieved by any deterministic algorithm for the *one-sided online bottleneck matching problem(OBMP)* is $\frac{k}{ln2} \approx 1.5k$ (k is the number of nodes in one side). The OBMP is different from our problem in two aspects. First, OBMP is a one-sided online problem. It assumes that all worker locations are static and known in advance, and tasks come one-by-one while we assume workers and tasks are both dynamically appearing. Second, OBMP does not allow buffered assignments. When a new request comes, OBMP needs to assign a worker to it immediately. While in our problem the assignment can be buffered to see if any better worker will come. Thus OBMP can be considered as a subproblem of ours. Next, we prove that the $\frac{k}{ln2}$ bound also holds for MMD-SC in Theorem II.1.

We first introduce two lemmas from [25] in facilitating the proof of Theorem II.1.

Lemma II.1. *For any k workers and k requests located on a horizontal real line, denote them as $\{w_1, w_2, ..., w_k\}$ and $\{r_1, r_2, ..., r_k\}$ by their locations in the left-to-right order. The matching*

$$\{(w_1, r_1), (w_2, r_2), ...(w_k, r_k)\}$$

is an optimal static min-max matching.

Lemma II.2. $2 + \frac{1}{2^{1/(k-1)}-1} \geq \frac{k}{\ln 2}$ *for $k \geq 2$.*

Then, we can have the theorem about the lower bound of competitive ratio for deterministic algorithms of MMD-SC.

Theorem II.1. *No deterministic algorithm for the* MMD-SC *problem with $k \geq 2$ workers can achieve a competitive ratio better than $\frac{k}{\ln 2}$.*

Proof. (Extended from the proof for online bottleneck matching in [25])

Given a real line space with the origin point $o = 0$, any location on it can be represented as the real number distance from o. Consider an adversary acts as below on the space:

1) On time $t_0 = 0$, let w_1 arrive at $l_1 = o$ and w_i arrive at $l_i = l_{i-1} + u^{i-1}$ for $i = 2, 3, ..., k$ and $u = 2^{\frac{1}{k-1}}$.
2) Also on time $t_0 = 0$, let r_1 arrive at $l_1 + 1$.
3) If w_1 has not be assigned, let r_i arrive at $l_i + 1$ on time t_i for $i = 2, 3, ..., k-1$, where $t_i = t_{i-1} + u^{i-1}$.
4) As soon as w_1 is matched to some r_j at time $t_h \leq t_x < t_{h+1}$(when r_h arrived but not r_{h+1}), let all rest requests arrive immediately: r_i at $l_{i+1} - u^j + 1$ for $i = h+1, h+2, ..., n-1$, and r_k at $l_1 - u^j + 1$.
5) If w_1 is still not matched after r_{k-1} arriving: hold until w_1 is the only unmatched one, then let r_k arrive at $l_k + 1$; otherwise the situation goes to 4.

Let OPT be the optimal static matching result and ALG be the best result of any deterministic algorithm.

If w_1 is matched to r_j before r_k arrives, r_k will be the left-most request. Because of Lemma 2.2, we know that $\{r_k, w_1\}$, $\{r_i, w_{i+1}\}$ is an optimal matching. So OPT=$u^j - 1$(max cost is from $r_{h+1}, ..., r_k$). In the mean time, ALG needs to spend the cost of matching w_1 to r_j: $\sum_{i=0..j-1} u^i$. So we have:

$$\frac{ALG}{OPT} \geq \frac{u^j - 1}{\sum_{i=0..j-1} u^i} = \frac{1}{u-1}$$

For the situation that w_1 is the last left-unmatched worker, $OPT = 1$ because each w_i is put at the right of r_i with distance 1. ALG has to match w_1 with r_k, then the cost is at least $1 + \sum_{i=1..n-1} u^i$. So we also have:

$$\frac{ALG}{OPT} \geq \frac{1}{\sum_{i=0..n-1} u^i} = \frac{1}{u-1}$$

Because of $u = 2^{\frac{1}{k-1}}$ and Lemma 2.3, $\frac{ALG}{OPT} \geq \frac{k}{\ln 2}$.

As the line space can be included in other Euclidean space or network space, in general, the competitive ratios achieved by any deterministic algorithms for MMD-SC are bounded by $\frac{k}{\ln 2}$ (k is the number of workers). □

Due to Theorem II.1, there is no deterministic algorithms can have a good enough performance for practical applications. To handle MMD-SC, we propose two deterministic algorithms in Section 3 and a random algorithm with a better theoretical bound in Section 4.

III. Two basic algorithms

In this section, we first propose two basic heuristic algorithms to handle the MMD-SC problem.

A. The Threshold Based Greedy Algorithm

We first propose a greedy algorithm for the MMD-SC problem, namely the *threshold based greedy* (TBG) algorithm, which matches any worker-request pair that has the travel time below a given threshold γ. In addition, TBG buffers new requests at most γ time if there are no available workers that can arrive at the required locations of the requests. For a new worker w_i, TBG assigns it with a request r_j that has been buffered more than γ time.

The intuition of TBG is that the threshold prevents bad cases where a request should be assigned to a worker who arrives a little later. The threshold should not be larger than the maximum cost because a worker arrives that long after the request will result in the maximum possible delay cost directly, so it is obvious not a good assignment.

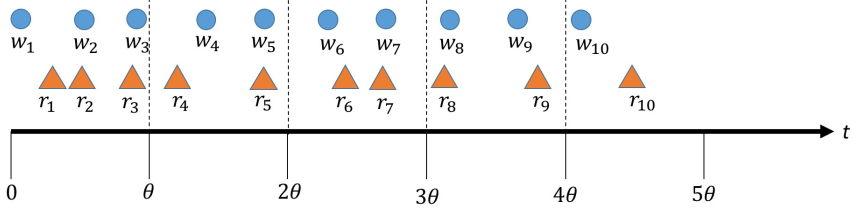


Fig. 2. A running example of Batch-based method for the MMD-SC problem.

TABLE I
ROUNDS OF THE RUNNING EXAMPLE OF BATCH-BASED METHOD

Timestamp	Available Workers	Requests	Assignment Triple
θ	$\{w_1, w_2, w_3\}$	$\{\}$	$\{\}$
2θ	$\{w_1, w_2, w_3, w_4, w_5\}$	$\{r_1, r_2, r_3\}$	$\{\langle w_1, r_3, 2\theta\rangle, \langle w_4, r_2, 2\theta\rangle, \langle w_5, r_1, 2\theta\rangle\}$
3θ	$\{w_2, w_3, w_6, w_7\}$	$\{r_4, r_5\}$	$\{\langle w_7, r_4, 3\theta\rangle, \langle w_2, r_5, 3\theta\rangle\}$
4θ	$\{w_3, w_6, w_8, w_9\}$	$\{r_6, r_7\}$	$\{\langle w_9, r_6, 4\theta\rangle, \langle w_8, r_7, 4\theta\rangle\}$
5θ	$\{w_3, w_6, w_{10}\}$	$\{r_8, r_9\}$	$\{\langle w_6, r_8, 5\theta\rangle, \langle w_{10}, r_9, 5\theta\rangle\}$
6θ	$\{w_3\}$	$\{r_{10}\}$	$\{\langle w_3, r_{10}, 6\theta\rangle\}$

Algorithm 1 The Threshold Based Greedy Algorithm

```
1:  procedure NEW-REQUEST($r_j$)
2:      if there is no available worker then
3:          push $r_j$ to the idle request queue $Q$
4:      else
5:          let $w_i$ be the nearest available worker to $r_i$
6:          if $d(l_i, l_j) \le \gamma$ then
7:              assign $w_i$ to $r_j$
8:          else
9:              push $r_j$ to the idle request queue $Q$
10:
11: procedure NEW-WORKER($w_i$)
12:     find the first request $r_j$ in the buffered queue $Q$ s.t.
        $d(l_i, l_j) + \bar{t} - t_j < \gamma$ or $\bar{t} - t_j > \gamma$ // $\bar{t}$ is the current
        timestamp
13:     if no such $r_j$ then
14:         set $w_i$ as available
15:     else
16:         assign $w_i$ to $r_j$
```

The detailed steps of TBG are described in Algorithm 1. The procedure *NEW-REQUEST* is invoked once a new request r_j comes. The procedure *NEW-WORKER* is invoked when a new worker w_i arrives.

The parameter γ is set to the largest possible assignment delay which can be estimated from the historical data. Unfortunately, the TBG algorithm does not have a good performance guarantee under the adversarial model [20], [31]. We will show the actual performance of it in Section V.

B. The Batch-Based Method

In the TBG algorithm, the server needs to respond to each task within λ time after it arrives, which, however, leaves very limited space for the server to select suitable workers and may prevent some later but more suitable workers from being selected. To alleviate the phenomena, we propose a batch-based method (BB), which buffers workers and tasks and periodically fires a batch process to match tasks that came in or before the last round with the current available workers. In other words, let the time interval between any two successive batches be θ, each coming task will be buffered at least θ, and then be assigned to a suitable worker in the next batch.

Algorithm 2 shows the details of BB. Specifically, it first sets the current timestamp as $\bar{t}$. Next, BB retrieves a set R' of requests arriving during the time period of $[0, \bar{t} - \theta]$ and a set W' of current available workers (lines 3-4). Then, BB utilizes the existing method for offline worker-and-task matching problems on R' and W' (line 5). Finally, the server notifies the selected workers to conduct their assigned requests (lines 6-7).

Algorithm 2 The Batch-Base Algorithm

```
1: procedure BATCHASSIGN
2:     let the current timestamp be $\bar{t}$
3:     let $R'$ be the set of requests that arrive in $[0, \bar{t} - \theta]$
4:     let $W'$ be the set of current available workers
5:     find the optimal assignment result $A'$ for $R'$ on $W'$
6:     for $\langle w_i, t_j, \bar{t}\rangle \in A'$ do
7:         notify worker $w_i$ to conduct task $t_j$
```

The value of θ can be set to the historical average assignment delay or other values smaller than that.

To better illustrate the batch-based method, we show the details of BB through a running example.

Example 2. *(Running example of the batch-based method) Assume there are 10 workers $w_1 \sim w_{10}$ (represented by blue circles) and 10 requests $r_1 \sim r_{10}$ (represented by yellow triangles). Their arriving sequence is shown in Figure 2. The results of batch-based method with the time interval of θ are shown in Table I. For example, at timestamp θ, the batch-based method is invoked for the first time, when there are three available workers $w_1 \sim w_3$, but no requests exist in time interval $[-\theta, 0]$. Thus, in the first round, no requests are handled. In the second round, there are five available workers $w_1 \sim w_5$. In addition, three requests $r_1 \sim r_3$ arrived during time interval $[0, \theta]$, then we use the existing offline algorithm [21] for the bottleneck maximum cardinality matching problem to decide the assignment triples as $\{\langle w_1, r_3, 2\theta\rangle$, $\langle w_4, r_2, 2\theta\rangle$, $\langle w_5, r_1, 2\theta\rangle\}$. In this way, we keep periodically invoking the* *batch-based method every θ time until all the requests are dispatched.*

The time complexity. Assume that, in each round, there are m available workers in the set W' and n requests in the set R', then the time complexity of Algorithm 2 is $O(((m+n)\log(m+n))^{\frac{1}{2}}(mn)^2)$. It needs $O(n)$ to retrieve a set of n requests in line 3 of Algorithm 2. Similarly, line 4 needs $O(m)$. To find the optimal assignment/matching for W' and R', it needs $O(((m+n)\log(m+n))^{\frac{1}{2}}(mn)^2)$ according to the results in [21]. In addition, since each request is assigned with just one worker, it needs $O(n)$ to send notifications in lines 6-7 of Algorithm 2. Thus, the time complexity of Algorithm 2 is $O(((m+n)\log(m+n))^{\frac{1}{2}}(mn)^2)$.

IV. An HST based solution

As discussed in Section II-C, no deterministic online algorithm for the MMD-SC problem can achieve a competitive ratio better than $O(n)$. Thus, we design an online random algorithm for a better theoretical performance guarantee. Existing works [11], [19] show that matching problems on tree structures usually can be solved better than those on general metric spaces. Inspired by this, we propose a Hierarchically well-Separated Tree (HST) based solution for the MMD-SC problem. Briefly, we first embed the given metric space into an HST and then solve the MMD-SC problem on the HST. For a n size space, the HST embedding causes $O(\log n)$ distortion, where the randomness is brought in during constructing the HST structure. We then design an algorithm with a constant 2 competitive ratio for the MMD-SC problem on HSTs. As a result, the entire solution can achieve a competitive ratio of $O(\log n)$, where n is the size of the space.

In the following, we first introduce the preliminary knowledge about HST and HST based space embedding in Section IV-A, next formally define the MDD-SC problem on HSTs in Section IV-B, then present our HST based algorithm with theoretical analyses in Section IV-C.

A. Preliminary

We first introduce the basic definitions and properties of HST, then show how to apply HST on metric space embedding in our algorithm.

1) Hierarchically well-Separated Tree: Briefly, HST is a kind of tree with a special structure which benefits the process of embedding general metric spaces into tree spaces. Below we give the formal definition of it and all necessary knowledges about HST based metric embedding that will be used in our algorithm. For more comprehensive introductions and theoretical details about HST, please refer to [19].

Definition 7 (k-HST). *A weighted tree T with root r is called a k-HST if it has the following properties:*

- *All leaves of T are at the same depth, i.e. at the bottom level;*
- *For any non-leaf node x, x has the same distance from all its children.*
- *For any non-leaf node x, the distance from x to its parent is always k times of the distance from x to any of its child.*

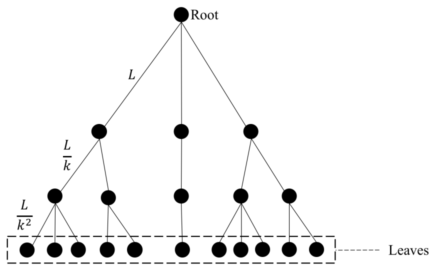


Fig. 3. An example of k-HST.

Algorithm 3 The Algorithm of building HSTs

```
1: procedure HST-PARTITION($(V = \{v_1, v_2, ..., v_n\}, d)$)
2:     Let $\pi(V)$ be a random permutation of $V$
3:     Choose a $\beta$ uniformly from $[1, 2]$
4:     Let $\delta$ be $\lceil \log_2 \Delta \rceil$ where $\Delta$ is the diameter of $V$.
5:     Init $D_\delta = V$ and $i = \delta - 1$
6:     while $D_{i+1}$ is not a singleton cluster do
7:         $\beta_i = 2^{i-1}\beta$
8:         for $l$ in $\pi(V)$ do
9:             for each cluster $S$ in $D_i$ do
10:                Create a new cluster of all unassigned vertices in $S$ closer than $\beta_i$ to $\pi(V)$
11:        $i = i + 1$
    return all clusters
```

Figure 2 gives the illustration of a k-HST with 4 levels and L as the weight in level edges. Next we give the basic concepts of metric embedding and how to do it with HST.

2) HST based metric embedding: Let (V, d) be a metric space, where V is a set of locations and d is the distance function of every pair of locations. Here $\mathcal{S}$ is a family of metric spaces over V, and $\mathcal{D}$ is a distribution over $\mathcal{S}$. We say that a metric (V, d') **dominates** (V, d), if $\forall u, v \in V$, $d'(u, v) > d(u, v)$. Also we say $(\mathcal{S}, \mathcal{D})$ **α-probabilistically approximates** (V, d), if all metrics in $\mathcal{S}$ dominate (V, d) and $\forall u, v \in V$, $E_{d' \in (\mathcal{S}, \mathcal{D})}[d'(u, v)] \leq \alpha \cdot d(u, v)$, where α is called the **distortion** of the approximated embedding.

For embedding with HST, we have the below theorem [19]:

Theorem IV.1. *Any metric space with size n can be $O(\log n)$-probabilistically approximated by a distribution over a family of HSTs.*

The steps to sample a 2-HST from a distribution is illustrated in Algorithm 3 (originally proposed by [19]).

Algorithm 3 takes a n size metric space (V, d) as input and embeds it into an 2-HST within the expected distortion $O(\log n)$. The main idea is to randomly decompose V from top to bottom and each cluster represents a node in the final HST (a trivial example is given in Figure 4). Its time complexity is $O(n^2)$.

The space of MMD-SC can be any types of metric space such as grid, Euclidean space or road network. We just show that all these spaces can be approximately embedded into an HST with the $O(\log n)$ distortion. The embedding also makes our solution randomized. Each point in the original space is

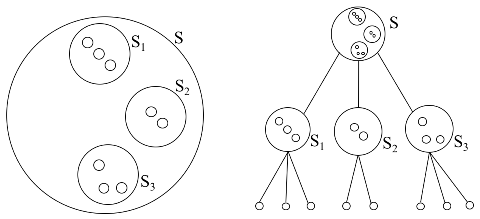


Fig. 4. An example of the decomposition in HST building. S is the first cluster that includes all vertices. S_1, S_2, S_3 are sub-clusters that may be created during the decomposition of S.

mapped to an HST leaf node. Because the tree construction is randomized, the adversary does not know the mapping thus cannot break the expected distortion. When n is large, the distortion cannot be ignored. Some preprocessing techniques can be used to reduce space size. Such as a) for a $n*n$ grid space, each k^2 neighboring grids can be merged as one grid then the space size is reduced to n^2/k^2 with an additional $2k$ distance deviation; b) for an Euclidean space area with radius r, a hexagonal partition with radius r_0 can transform it into a $\frac{2\pi r^2}{\sqrt{3}r_0^2}$ size graph [22]; c) for road networks, the k-shortest-path cover technique mentioned in [16] can be used.

B. The MMD-SC Problem on HST

After the space embedding, the original MMD-SC problem becomes a new matching problem on HST as follows:

Definition 8. *(MMDH) Given an HST $T = (\alpha, V, d)$, a set of workers W on the leaves of T and a set of requests R with $|R| = |W|$. MMDH is to find a set of assignment triples A between R and W s.t. $\max_{a_{ijx} \in A} cost(a_{ijx})$ is minimized.*

Different from the MMD-SC problem, the MMDH problem has some unique features due to the HST structure. The most significant difference is the possible locations of requests and workers. Because HST embedding maps points in the original metric space only to leave nodes of HST, requests and workers can only appear at leaf nodes in MMDH. By utilizing the tree structure and the leaf-nodes-only feature of HST, we designed a algorithm for MMDH with a constant 2 competitive ratio.

Before presenting our algorithm for MMDH, we first discuss two special cases of the problem: 1) the "offline" case when both workers and requests have arrived; 2) the "one-side online" case when workers have arrived and requests come one-by-one.

The Offline Case. This case is identical to the problem of the static version of MMDH, i.e. given all requests and workers in advance and match them with the min-max goal. Due to the tree structure, this problem can be solved by a simple greedy method, namely *level by level greedy* (LLG) algorithm, shown in Algorithm 4, which simply matches workers and requests from the bottom level to the upper level one by one.

Specifically, LLG iterates all subtrees from height 0 (leaf nodes) to max height (the whole tree), match all worker-request pairs of the current subtree in any order. Obviously the last matched pair gives the maximum distance.

Algorithm 4 The Level by Level Greedy Algorithm

```
1: procedure MATCHING($T, S, R$)
2:     for $l$ from 0 to the height of $T$ do
3:         for each subtree $t$ of level $l$ do
4:             match all worker-request pairs in $t$
```

Similar to the offline min sum matching on HST [11], LLG also gives the optimal solution to offline min max matching, as shown below:

Lemma IV.1. *The last matched pair in LLG is the optimal min max matching pair.*

Proof. For any i level subtree T_i iterated by LLG, its own subtrees are also iterated from height 0 to max height. The iteration can be considered as LLG removes worker-request pair from it batch by batch and remove all possible pairs during the max height iteration. If T_i has same numbers of workers and requests (i.e. perfect matchings exist in it), one of them must be found on or before LLG finishes the i level iteration.

Let $d(s, r)$ be the max distance found by LLG at a h level subtree T_h. If there exists a matching M' s.t. the max distance $d(s', r') < d(s, r)$, then M must match all workers and requests below h level. That means there exists a perfect matching among all $h-1$ level subtrees of T_h. This is not possible because LLG cannot miss such a perfect matching. □

The One-Side Online Case Following the convention of online bipartite matching problems [39], [40], when worker locations are given and requests come one by one, the problem is considered to be a *one-side online* problem. We find that the one-side online MMDH case can be solved by the naive greedy algorithm (Greedy): always match the new coming request r with the nearest worker, if multiple nearest workers exist, choose arbitrarily.

Lemma IV.2. *Greedy gives the optimal solution for the one-side online MMDH problem.*

Proof. Suppose d_{max} on level h is the max distance achieved by Greedy and (l_i, l_j) is the first matched pair s.t. $d(l_i, l_j) = d_{max}$. If there is another matching M' between R, W that has the max distance $d'_{max} < d_{max}$, then we know the matched distance of r_j in M' must be smaller than $d(l_i, l_j)$ (i.e., $d(l_j, M'(r_j)) \le d'_{max} < d(l_i, l_j)$). In other words, M' must match r_j inside a subtree $T_{<h}$ below level h.

Let the $h-1$ level subtree containing r_j be T_{h-1}, then $T_{<h}$ is a subtree of T_{h-1}. When Greedy assigns r_j to w_i, all workers in T_{h-1} must have been already assigned, because w_i is currently the nearest worker to r_j. Before r_j arriving, there are exactly the same number of requests and workers inside T_{h-1}. On the other hand, M' assigns one of the workers in T_{h-1} to r_j, which means one of these earlier arrived requests must be assigned outside T_{h-1}. Thus, the maximum distance of M' is at least on level h (i.e., $d'_{max} \ge d_{max}$). The contradiction indicates no such matching exists. □

Algorithm 5 The hold procedure

```
procedure HOLD($w_i$, $r_j$)
    set $w_i$ to be held by $r_j$
    put $r_j$ to the holder queue
    init a count down timer $timer(r_j) = d(l_i, l_j)$
    call POP-HOLDER on $timer(r_j)$ reaches 0
procedure POP-HOLDER
    let $r_j$ be the head of the holder queue
    if $timer(r_j) \leq 0$ then
        pop $r_j$ from the holder queue
        assign $r_j$ to the worker hold by it
        call POP-HOLDER
```

C. The Algorithm for MMDH

The buffer-allowed setting of MMD-SC and MMDH permits that the system needs not to immediately match, i.e. when there are both unserved requests and available workers, we can leave both of them waiting for better opposite peers instead of matching them at once.

The main idea of our algorithm (Algorithms 5 and 6) is to give the earlier arrived request the best possible worker by buffering it for a limited time period. The key points of the algorithm are how to determine which worker is the best one and how long should it be buffered. We propose different statuses of requests and workers for better illustrating. Arrived and not assigned requests have two possible statuses: "idle" and "holding". Each status has a first-in-first-out queue that stores requests the corresponding status following their arriving order. Workers also have two statuses: "available" or "being held by a request". A holding request will not go back to idle. A worker being held by a request may become available again if the request releases him/her and holds some other worker.

Before presenting the major matching algorithm, we first show the new *hold* operation which is used to virtually reserve a worker to a request. Specifically, a worker w_i is held by a request r_j means w_i is the "best" candidate worker for r_j currently and the assignment will be performed after a certain buffering time period if no better workers arrives. All holding request-worker pairs are in a global First-In-First-Out holder queue. When a pair is pushed to the queue, a counting-down timer is initiated in the meanwhile. Once a request is pushed, its status will be *holding* until it becomes the head of the queue and its timer has also ended. The procedure of *hold* is illustrated in Algorithm 5.

Next we show an algorithm utilizing the hold operation. Similar as the Threshold Based Greedy algorithm, it is also composed by two procedures that handle newly coming requests and workers respectively.

When a new request r_j arrives, if there is no available worker, it will be put at the tail of the idle queue. Otherwise, let it hold the nearest available worker w_i and put it at the tail of the holding queue. At the same time, initiate a counting down timer with value of $d(l_i, l_j)$ (i.e., the travel cost between w_i, r_j, recalling that in MMD-SC, the travel cost and delay cost are considered the same). During the holding time period, w_i may be replaced by a better worker w'_i and the timer may be reseted to a smaller value of $d(l'_i, l_j)$. When the timer goes to 0 and r_j is the head of the holding queue (i.e. r_j is the earliest arrived request not assigned yet), assign the holding worker to r_j.

Algorithm 6 The Algorithm for MMDH

```
procedure NEW-REQUEST($r_j$)
    if there is no available workers then
        push $r_j$ to the idle request queue.
    else
        let $w_i$ be the nearest available worker to $r_j$ (choose arbitrarily if multiple exist)
        HOLD($w_i, r_j$)

procedure NEW-WORKER($w_i$)
    for each $r_j$ in the holder queue do
        let $w_{r_j}$ be the worker held by $r_j$
        if $timer(r_j) > d(l_i, l_j)$ then
            release $w_{r_j}$ and set $w_i$ to be held by $r_j$
            reset $timer(r_j)$ to $d(l_i, l_j)$
            NEW-WORKER($w_{r_j}$)
            return
    if the idle request queue is not empty then
        let $r_j$ be the head of the idle request queue
        HOLD($w_i, r_j$)
    else
        set $w_i$ as available
```

When a new worker w_i arrives, we first check if it is a better worker for any of the holding requests in the arrival time priority. If so, release the previous holding one w_{r_j} and hold w_i instead, then consider w_{r_j} as a newly coming worker. w_i is considered better than w_{r_j} if and only if $timer(r_j) > d(l_i, l_j)$ (i.e., in the current timestamp, the cost between r_j and w_i is less than that between r_j and w_{r_j}). Otherwise, let the first request in the idle queue hold w_i.

Theorem IV.2. *The competitive ratio of Algorithm 6 for MMDH is 2.*

Proof. For a request r_j, w'_i is considered better than w_i if $d(w_i, r_j) + max(t_{w_i} - t_{r_j}, 0) \geq d(w'_i, r_j) + max(t_{w'_i} - t_{r_j}, 0)$. We first show that the algorithm gives each request the best possible worker (although not assign immediately) with the arriving time priority.

If w_{r_0}, the best worker of the first arriving request r_0, arrives earlier than r_0, it will be held by r_0 at the beginning and will not be released. If w_{r_0} arrives later than r_0, it will be held just on its arriving and also will not be released (because r_0 has the highest priority). So r_0 gets its best worker. For a later arriving request r_i and its best worker be w_{r_i}, there are several situations: a) w_{r_i} is also the best worker of some previous request r_{i-x}; b) w_{r_i} comes earlier than r_i and never held by earlier arriving requests; c) w_{r_i} comes earlier and has been held by earlier arriving requests; d) w_{r_i} comes later than r_i.

Situation a) should not be considered because r_{i-x} has a higher priority than r_i. Situation b) and d) is the same as w_{r_0}

TABLE II
SYNTHETIC LOCATION DATA GENERATION SETTING

name	distribution	parameters	size
L1	Uniform	None	1000*1000
L2	Normal	$\mu = 500, \sigma = 50$	1000*1000

TABLE III
SYNTHETIC ARRIVING TIMES GENERATION SETTING

name	distribution	parameters	t_{max}
T1	Uniform	None	2000
T2	Zipf	a=2	2000
T3	Normal	$\mu = 1000, \sigma = 200$	2000

for r_0. For situation c), assume the earlier arriving request always get its own best worker, then w_{r_i} will be released finally. Because r_0 can get its best worker, so by induction, all requests get their best worker.

If all assignments are done immediately, i.e. no waiting time cost, the min-max cost will be the same as the optimal min-max cost of a one-side online case with each distance $d'(r, w) = d(r, w) + max(t_w - t_r, 0)$. We have shown that the naive greedy algorithm gives the best solution for the offline cases in Lemma IV.1 and Lemma IV.2. Because each request gets its best worker, now we only need to check the relation between the waiting time $waiting(r_i)$, the travel time $travel(r_i)$ and the assignment cost $cost(r_i)$.

For r_0, its waiting time cannot be larger than its best cost, thus not larger than the optimal min max cost OPT_{minmax}. Furthermore, r_0's waiting time is at least the same as its travel cost, so we have:

$$cost(r_0) \leq 2 \cdot waiting(r_0) \leq 2 \cdot OPT_{minmax}$$

For later coming requests, the situation is the same as r_0 if they are not blocked. For any r_i blocked by r_{i-1}, either

$$waiting(r_i) \leq waiting(r_{i-1}) \leq OPT_{minmax}$$

or

$$waiting(r_i) \leq travel(r_i) \leq OPT_{minmax}$$

For either case we have $cost(r_i) \leq 2 \cdot OPT_{minmax}$. □

Remember that the HST based metric embedding brings the $O(\log n)$ expected distortion. By combining the HST embedding and Algorithm 6, we get the online random algorithm for MMD-SC with a competitive ratio of $O(\log n)$, where n represents the size of the embedded metric space.

V. EXPERIMENTS

In this section, we study the performance of all three algorithms for the MMD-SC problem on a taxi trip dataset and a synthetic dataset.

A. Experiment Setup

In this part, we first introduce the real and synthetic datasets then present our detailed evaluation methods and goals.

1) Real Datasets: We use the taxi trip data in New York city from NYC Taxi and Limousine Commission [6]. We choose the data of yellow taxi in Jan 2017 and Feb 2017. We assume that passengers are task requesters and taxis are crowd workers. The taxi trip data contains the time and location information of pick-ups and drop-offs. We use the pick up timestamps and locations as the timestamps and locations of requests, respectively. In addition, we use the drop off location to initialize the location of the worker because it is where a taxi becomes available again. Although one taxi may appear multiple times in the overall period, each drop off is considered as an independent worker.

2) Synthetic Datasets: For all synthetic datasets, request and worker locations are sampled from the space with different distributions. The detail parameters are shown in the Table II. For the HST based method, we first build the HST for each working space and then run the algorithm on it.

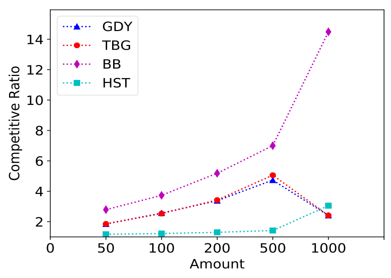
For the arriving time, we assume all requests and workers arrive at discrete timestamps sampled from $\{0, 1, 2, ..., t_{max}\}$. Each virtual time span $[t_0, t_1)$ in this setting can be considered as a $t_1 - t_0$ seconds in real world time. The detail settings are presented in Table III. For each synthetic data setting, to reduce the randomness of sampling, we run experiments for 20 times and report the average results.

3) Evaluation Methods and Goals: Other than our proposed algorithms, including the batch based method in Section III-B (BB), the threshold based greedy algorithm in Section III-A (TBG) and the *HST* based algorithm in Section IV-C (HST), we also use a naive greedy algorithm (GRY) as a baseline. GRY can be considered as TBG with the threshold of 0.

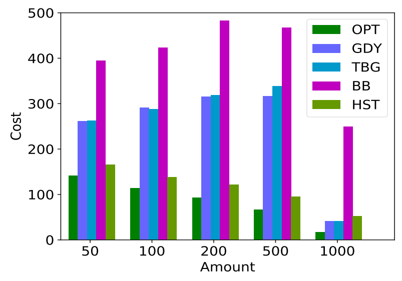
The primary goal of the experiments is to compare the effectiveness of different algorithms. As mentioned in Section II, the optimal solution of our problem is identical to the result of the best offline min-max matching problem with the matching cost being the summation of the travel cost pluses the minimum delay cost. Thus, we first compute the optimal solution OPT with the offline algorithm described in [13]. Then, to get a universal comparison between different datasets, the performance of an online algorithm A on a dataset D represented as $\frac{A(D)}{OPT(D)}$ (i.e., the actual competitive ratio), where $A(D)$ and $OPT(D)$ are the maximum delays of the results achieved by algorithm A and OPT respectively. In addition, we report the average of all delays for each algorithm to compare their performance for the min-sum goal.

We also try to find out how the following parameters affect the performances.

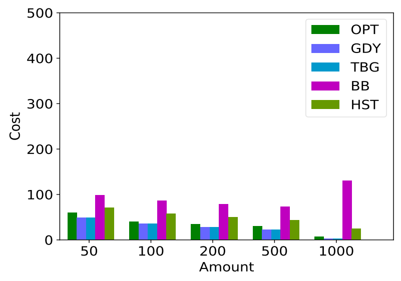
- **Density of workers and requests.** We use density to represent the number of workers and requests within a fixed area and time period. Spatial crowdsourcing tasks of different kinds or in different time may have quite different task/worker densities. For example, the total taxi trips in a metropolis [6] are usually more than tens of thousands while the number of express tasks [4] may be less than one thousand. Usually the performance of matching problems is stable when the density increasing (e.g., the total travel cost in [41]). While the matching



(a) Competitive Ratio

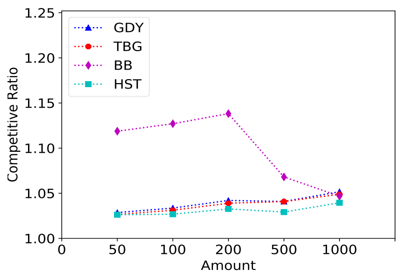


(b) Maximum Cost

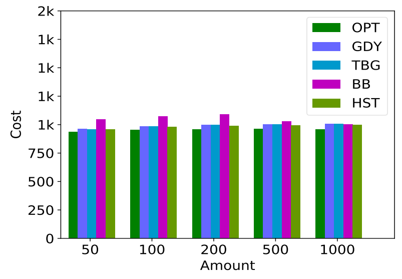


(c) Average Cost

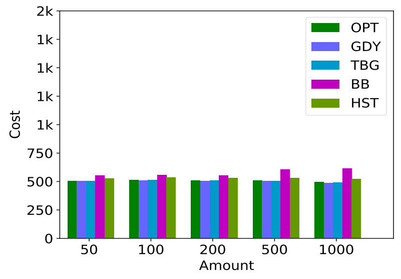
Fig. 5. Effects of density of workers and requests.



(a) Competitive Ratio



(b) Optimal Max Cost



(c) Average Cost

Fig. 6. Results of Zipf distributed data.

delay, a major distinction between our problem and existing ones, will be affected by the temporal density intuitively. When the temporal density is relatively small, the delay cost will be the major part of a matching. Thus we vary both spatial and temporal density together and vary temporal density alone in our synthetic dataset to present their effects.

- **Distribution of workers and requests.** Spatial crowdsourcing requests, for both arriving timestamps and locations, may have quite different distributions in different situations. For example, the taxi calling service usually has several peak hours and package delivery service has a much more stable and uniform distribution during a whole day. Thus we check the effects of different types of distributions as shown in Table II and III .
- **Size of HST.** For the HST-based algorithm, the HST is considered as a pre-built input data structure. For the space with various or even unlimited vertices such as a Euclidean space or a grid space, the vertex amount is too large for an HST. Then each HST node will be used to represent a bunch of original vertices or an area. Then the location errors between the real vertex and the HST node are introduced other than the embedding distortion. A larger HST size will give a smaller location error but have a larger embedding distortion. Thus we pre-build HSTs with different sizes to check this effect and the result is shown in Figure 10(a).

B. *Results on Synthetic Datasets*

1) Effect of density of workers and requests: We first study the effect of the data density by varying the total numbers of workers and requests in a fixed space and time span. We use L1 and T1 settings for both workers and requests generation and vary data size in $\{50, 100, 200, 500, 1000\}$, as shown in Figure 5.

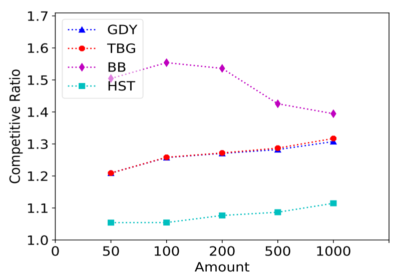
From Figures 5(a) and 5(b), we observe that HST outperforms other algorithms in most cases. Intuitively, more requests and workers bring smaller matching costs. This effect is confirmed by the decrease of the optimal results in Figure 5(b). When there are 1000 workers and requests, the data density is high enough for both GDY and TBG to find a good matching easily. But the tree size of HST does not change, the location errors become non-negligible. In addition, since we use a fixed batch size for BB, the performance of it becomes worse when the data density increases.

Figure 5(c) shows that HST has a larger average matching cost than OPT. The reason is that the *hold* procedure in our HST algorithm brings a waiting cost for all matchings. When the data density is high, this waiting cost becomes the major cost. Also we can see that both GDY and TBG have a better performance than OPT in this part. This is because the offline bottleneck matching algorithm needs to consider the global maximum cost and sacrifices some of the total cost while GDY and TBG just find the current minimal matching.

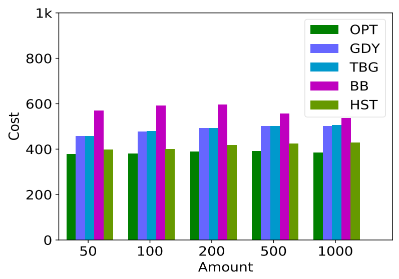
2) Effects of distribution of workers and requests: In this part we vary different types of distributions for both workers and requests. The following combinations are tried: 1) L1, T1 for workers and L1, T2 (Zipf distribution) for requests; 2) L1, T1 for workers and L2 (Normal distribution), T3 (Normal distribution) for requests. The results are shown separated in Figure 6 and Figure 7.

Figure 6 shows that all algorithms perform well (all competitive ratios are below 1.2). This is because most requests come at the beginning, then they need to wait for a long time to be matched. In addition, the density does not affect the performance because the delay cost is the major cost here.

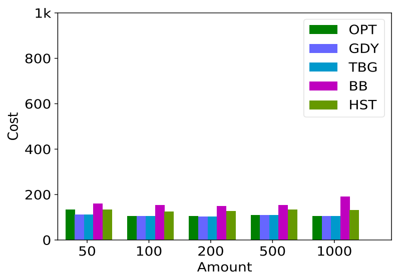
In Figure 7, the locations and the arriving timestamps of requests both follow a Normal distribution. Most requests come at the center of the space and in the middle of the overall



(a) Competitive Ratio

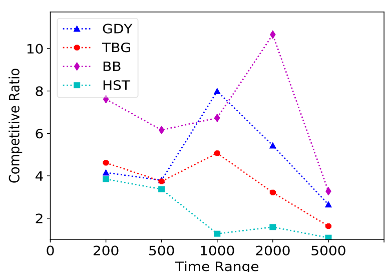


(b) Optimal Max Cost

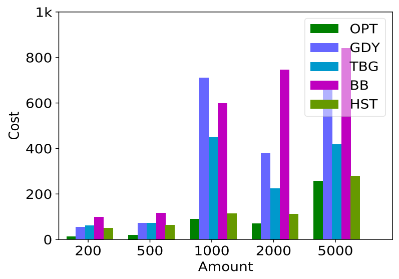


(c) Average Cost

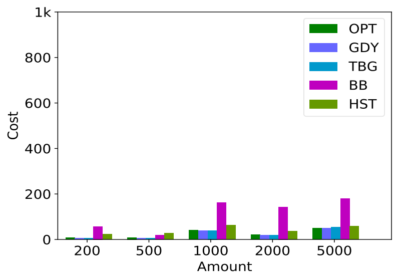
Fig. 7. Results of Normal distributed data



(a) Competitive Ratio



(b) Maximum Cost



(c) Average Cost

Fig. 8. Effects of relative density difference between spatial and temporal.

time span. Similar to the Zipf case, there are always some requests who need to wait for a long time, thus all algorithms perform not bad. We can also observe that the competitive ratio of HST is lower than other tested algorithms.

3) Effects of relative density difference between spatial and temporal: We use L1, T1 for workers and use L1 , T3 for requests and vary the time span of both requesters and workers in $\{[0, 200), [0, 500), [0, 1000), [0, 2000), [0, 5000)\}$ with the size fixed to 1000. That is, we try the same spatial density with different temporal densities.

From Figure 8(a) we can see that the relative density affects the performance a lot. When the temporal density is high (time span $\{[0, 200), [0, 500)\}$), HST has no advantages to GDY and TBG. This is because there are enough new arriving workers to be matched and the delay cost can be ignored. When the temporal density decreases, HST begins to outperform other methods.

C. Results on Real Datasets

We test the proposed algorithms on each daily data and use the average result as the final result. For each daily data, we group them by hours and show the result of some representative hours in Figure 9. The taxi trip dataset does not have any obvious patterns or distributions. The major difference between different hours is the data density. 6PM has around 5000 trips and 2AM has less than 1000.

In Figure 9, the taxi trip data does not have a peak of densities. HST also has the best competitive ratios among all methods. In Figure 9(c), GDY and TBG perform better than BB and HST because both BB and HST have some compulsive waiting time for all requests.

In addition, we sample one group of taxi trip data in 6 PM to evaluate the parameters of our algorithms. The result is shown in Figure 10. We vary the size of HST, the size of one batch, the size of threshold and show the result in Figures 10(a), 10(b) and 10(c). We can see that all these parameters have effects on the performance especially for the TBG and BB. The size of HST has a smaller effect but the increasing trend is obvious.

D. Summary of Experiment Results

We summarize our major findings as follows:

- HST has a better and more stable competitive ratio compared to all other algorithms in most cases.
- When the offline optimal cost is large, simple heuristic algorithms GDY and TBG have good competitive ratios.
- The size of HST has a limited effect on the performance.

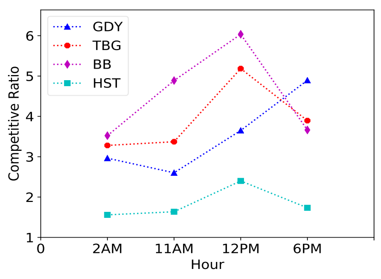
VI. Related Works

In this section, we review related works from two categories, task assignment for spatial crowdsourcing and online matching problems.

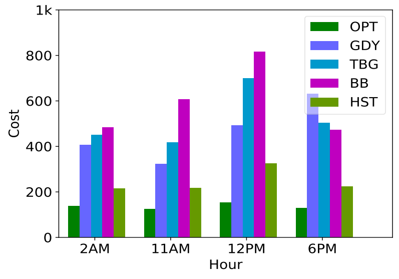
A. Task assignments for Spatial Crowdsourcing

In recent years, with the fast development of smartphones and other mobile devices, spatial crowdsourcing becomes more and more popular in various applications such as Offline-to-Online (O2O) services and online taxi order services. Task assignment is one the the core problems in spatial crowdsourcing [33], [34], [36]–[39], [41], [43]–[45].

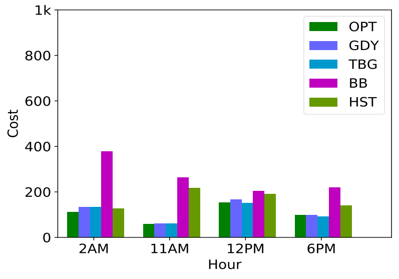
To our best knowledge, [27] first proposed the task allocation problem on spatial crowdsourcing. They try to maximize the platform's throughput in with batch-based algorithms, i.e. the total number of assigned tasks. Some follow up works also focused on how to do better batch-based assignment for spatial tasks and propose additional constraints and goals, such as [28] introduced the crowd worker reliability, [36] proposed the maximum assigned utility goal and [15] added an additional spatial temporal diversity goal. Privacy issues in spatial task assignment are also studied in [32], [35]. All of them consider the task assignment as an static matching problem aiming to maximize the total throughput.



(a) Competitive Ratio

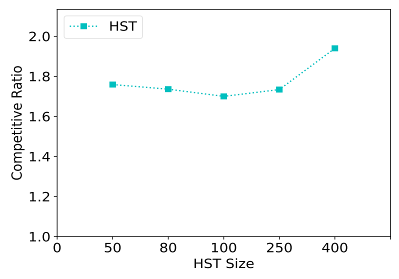


(b) Maximum Cost

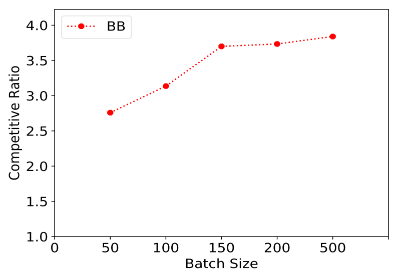


(c) Average Cost

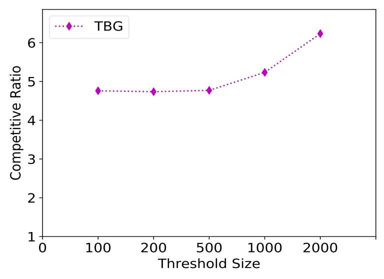
Fig. 9. Results for real data.



(a) Competitive Ratio



(b) Competitive Ratio



(c) Competitive Ratio

Fig. 10. Effects of parameters.

In the real world scenario, both workers and tasks come dynamically. Thus the spatial task assignment is essentially an online problem. [24] first used an online model to describe the assignment process and proposed a novel method for one-side online task assignment. [39] first formalized a two-side online matching problem with the total utility maximization goal and proposed well-performed online algorithm under the random order evaluation model. Then [41] further generalized the online model and provided an assignment algorithm with better performance under i.i.d evaluation model (both tasks and workers). In addition, [33] studied the trichromatic task assignment problem. Unlike the traditional bipartite worker-task matching, they considered the matching with three parties: workers, requesters and places of tasks.

B. Online matching problems

As one of the fundamental problems of combinatorial optimization, various different online versions of matching/ bipartite matching problems have been well studied in past decades. In this part we review some representative works of them that are related to our problem.

1) Min Max Matching Problems: The min max matching problem, also known as bottleneck assignment or worst case matching problem, was firstly proposed by [23]. A well-known solution for it is *Threshold* [13] which searches the bottleneck by solving multiple max cardinality matching problems with high cost edges removed from the input bipartite graph. Based on *Threshold*, [21] proposed an improved algorithm with a faster bottleneck searching process. Also [17] showed the problem can be solved more efficiently on some special spaces. Recently, [30] proposed a variant with additional capacity constraints and gave an algorithm with good efficiency and scalability on Euclidean spaces. The Online Bottleneck Matching problem mentioned in Section II can be considered as an one-sided online extension of these works. The major differences between our problem and them is that we have a two-sided online setting and the delay time goal.

2) Online Weighted Bipartite Matching Problems: [26] and [29] first studied the problem of online weighted matching respectively. Their problem can be briefly described as an one-side online variant of the traditional min-sum weighted bipartite matching problem [13]. They showed that the performance bound for any deterministic algorithm is $2k-1$ where k is the number of nodes in one side of the bipartite graph and gave an algorithm known as *PERMUTATION* that archived this bound. [26] also mentioned that *PERMUTATION* can archive the $2k-1$ bound for the min-max goal.

3) Online Matching With Delays Problems: In the original online matching problems, assignment decisions have to be made immediately. In other words, delayed matchings are not allowed. Some very recent works [8]–[10], [18] studied the online matching problem with delays.

[18] first proposed the problem of *mincost perfect matching with delays* (MPMD). Briefly, given a sequence of requests on a metric space, MPMD tries to find a min-sum matching between these requests with both the distance cost and the delay cost . A typical application of MPMD is the online two-player game matching where the platform tries to minimize both the waiting time before players are matched and the skill difference between each matched pair. [8] and [10] then extended MPMD to *the min-cost bipartite perfect matching with delays* (MBPMD) problem by adding a polarity property (either positive or negative) of the requests to be matched. The goal of MBPMD is to minimize the total distance cost and delay cost but it only allows matching between requests with different polarities. In other words, MPMD is about matchings over general graphs and MBPMD is about bipartite graphs. [8] showed the competitive ratio of any randomized algorithm for MBPMD is $\Omega(\sqrt{\frac{\log n}{\log\log n}})$. They also provided an $O(\log n)$-competitive random algorithm and a $(10h)$-competitive deter-

ministic algorithm on trees of height h.

MBPMD is the most related existing work to our problem. There are two major differences. First, their goal is to minimize the total cost of all matched pairs and our goal is to minimize the maximum matching cost. Theoretically min-sum matching problems and min-max matching problems have quite different combinatorial properties [13]. Second, MBPMD considers the delay cost of both bipartite sides and our problem considers only the waiting time of task requesters. Their setting is reasonable for those applications where two parties are not essentially different, such as online game players. While our setting is based on the spatial crowdsourcing background where the experience of requesters is the critical indicator.

VII. Conclusion

In this paper, we propose the problem of minimizing the maximum delay in spatial crowdsourcing (MMD-SC). Particularly, the workers and requests keep coming on a metric space and we need to assign each request a suitable arrived worker dynamically and minimize the maximum request delay time. We show that deterministic algorithms for the MMD-SC problem have a lower bound $\frac{k}{\ln 2}$ of competitive ratio, where k is the number of total workers. We propose two heuristic algorithms, namely the threshold based algorithm and the batch-based algorithm, and one Hierarchically well-Separated Tree (HST) based online random algorithm on MMD-SC, which has a competitive ratio $O(\log n)$ (i.e., the expected maximum cost of our HST-based algorithm is at most $O(\log n)$ times to the optimal maximum cost). Extensive experiments have shown the efficiency and effectiveness of our proposed algorithms on both real and synthetic data sets.

Acknowledgment

The work is partially supported by RGC/NSFC N_HKUST637/13, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Hong Kong ITC ITF grants ITS/425/16FX and ITS/212/16FP , and Microsoft Research Asia Collaborative Research Grant. Peng Cheng and Lei Chen are the corresponding authors in this paper.

References

[1] Didi chuxing. https://www.didichuxing.com.
[2] Ele.me. http://ele.me.
[3] Gigwalk. https://gigwalk.com.
[4] Gogovan. https://www.gogovan.com.hk/en/.
[5] Taskrabbit. https://www.taskrabbit.com.
[6] Trip record data from nyc taxi and limousine commission. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.
[7] Uber. https://www.uber.com.
[8] I. Ashlagi, Y. Azar, and et al. Min-cost bipartite perfect matching with delays. In *LIPIcs*, volume 81, 2017.
[9] I. Ashlagi, Y. Azar, and et al. Min-cost matching with delays. 2017.
[10] Y. Azar, A. Chiplunkar, and H. Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *SODA*, pages 1051–1061. Society for Industrial and Applied Mathematics, 2017.
[11] N. Bansal, N. Buchbinder, and et al. An o (log 2 k)-competitive algorithm for metric bipartite matching. *Algorithms–ESA 2007*, pages 522–533, 2007.
[12] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.
[13] R. E. Burkard, M. Dell'Amico, and S. Martello. *Assignment problems, revised reprint*, volume 125. Siam, 2009.
[14] P. Cheng, X. Lian, L. Chen, and C. Shahabi. Prediction-based task assignment in spatial crowdsourcing. In *ICDE*. IEEE, 2017.
[15] P. Cheng, X. Lian, Z. Chen, et al. Reliable diversity-based spatial crowdsourcing by moving workers. *PVLDB*, 8(10):1022–1033, 2015.
[16] P. Cheng, H. Xin, and L. Chen. Utility-aware ridesharing on road networks. In *SIGMOD*, pages 1197–1210, 2017.
[17] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.
[18] Y. Emek, S. Kutten, and R. Wattenhofer. Online matching: haste makes waste! In *STOC*, pages 333–344. ACM, 2016.
[19] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC*. ACM, 2003.
[20] J. Feldman, M. Henzinger, and et al. Online stochastic packing applied to display ad allocation. In *ESA*, pages 182–194. Springer, 2010.
[21] H. N. Gabow and R. E. Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9(3):411–417, 1988.
[22] A. Gorodnik and A. Nevo. Counting lattice points. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 2012.
[23] O. Gross. The bottleneck assignment problem. Technical report, RAND CORP SANTA MONICA CALIF, 1959.
[24] U. U. Hassan and E. Curry. A multi-armed bandit approach to online spatial task assignment. In *UTC-ATC-ScalCom*. IEEE, 2014.
[25] R. Idury and A. Schaffer. A better lower bound for on-line bottleneck matching, 1992.
[26] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.
[27] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *SIGSPATIAL*. ACM, 2012.
[28] L. Kazemi, C. Shahabi, and L. Chen. Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In *SIGSPATIAL*. ACM, 2013.
[29] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.
[30] C. Long, R. C.-W. Wong, and et al. On optimal worst-case matching. In *SIGMOD*, pages 845–856. ACM, 2013.
[31] A. Mehta et al. Online matching and ad allocation. *FnT-TCS*, 8(4):265–368, 2013.
[32] L. Pournajaf, L. Xiong, V. Sunderam, and S. Goryczka. Spatial task assignment for crowd sensing with cloaked locations. In *MDM*, volume 1, pages 73–82. IEEE, 2014.
[33] T. Song, Y. Tong, and et al. Trichromatic online matching in real-time spatial crowdsourcing. *ICDE*, pages 1009–1020, 2017.
[34] H.-F. Ting and X. Xiang. Near optimal algorithms for online maximum edge-weighted b-matching and two-sided vertex-weighted b-matching. *Theoretical Computer Science*, 607:247–256, 2015.
[35] H. To, G. Ghinita, and C. Shahabi. A framework for protecting worker location privacy in spatial crowdsourcing. *PVLDB*, 7(10):919–930, 2014.
[36] H. To, C. Shahabi, and L. Kazemi. A server-assigned spatial crowdsourcing framework. *ACM TSAS*, 1(1):2, 2015.
[37] Y. Tong, L. Chen, and C. Shahabi. Spatial crowdsourcing: Challenges, techniques, and applications. *PVLDB*, 10:1988–1991, 2017.
[38] Y. Tong, L. Chen, Z. Zhou, H. V. Jagadish, L. Shou, and W. Lv. SLADE: A smart large-scale task decomposer in crowdsourcing. *IEEE Trans. Knowl. Data Eng.*, 30(8):1588–1601, 2018.
[39] Y. Tong, J. She, and et al. Online minimum matching in real-time spatial data: experiments and analysis. *PVLDB*, 9(12):1053–1064, 2016.
[40] Y. Tong, J. She, and et al. Online mobile micro-task allocation in spatial crowdsourcing. *ICDE*, pages 49–60, 2016.
[41] Y. Tong, L. Wang, and et al. Flexible online task assignment in real-time spatial data. *PVLDB*, 10:1334–1345, 2017.
[42] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye. Dynamic pricing in spatial crowdsourcing: A matching-based approach. In *SIGMOD*, pages 773–788, 2018.
[43] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu. A unified approach to route planning for shared mobility. *PVLDB*, 11(11):1633–1646, 2018.
[44] Y. Tong and Z. Zhou. Dynamic task assignment in spatial crowdsourcing. *SIGSPATIAL Special*, 10(2):18–25, 2018.
[45] Y. Zeng, Y. Tong, L. Chen, and Z. Zhou. Latency-oriented task completion via spatial crowdsourcing. In *ICDE*, pages 317–328, 2018.