Online Minimum Matching in Real-Time Spatial Data: Experiments and Analysis

Yongxin Tong [†] Jieying She [§] Bolin Ding [‡] Lei Chen [§] Tianyu Wo [†] Ke Xu [†]
[†]SKLSDE Lab, NSTR, and IRI, Beihang University, China
[§]The Hong Kong University of Science and Technology, Hong Kong SAR, China
[‡]Microsoft Research, Redmond, WA, USA
[†]{yxtong,woty,kexu}@buaa.edu.cn, [§]{jshe,leichen}@cse.ust.hk, [‡]bolind@microsoft.com

ABSTRACT

Recently, with the development of mobile Internet and smartphones, the online minimum bipartite matching in real time spatial data (OMBM) problem becomes popular. Specifically, given a set of service providers with specific locations and a set of users who dynamically appear one by one, the OMBM problem is to find a maximum-cardinality matching with minimum total distance following that once a user appears, s/he must be immediately matched to an unmatched service provider, which cannot be revoked, before subsequent users arrive. To address this problem, existing studies mainly focus on analyzing the worst-case competitive ratios of the proposed online algorithms, but study on the performance of the algorithms in practice is absent. In this paper, we present a comprehensive experimental comparison of the representative algorithms of the OMBM problem. Particularly, we observe a surprising result that the simple and efficient greedy algorithm, which has been considered as the worst due to its exponential worst-case competitive ratio, is significantly more effective than other algorithms. We investigate the results and further show that the competitive ratio of the worst case of the greedy algorithm is actually just a constant, 3.195, in the average-case analysis. We try to clarify a 25-year misunderstanding towards the greedy algorithm and justify that the greedy algorithm is not bad at all. Finally, we provide a uniform implementation for all the algorithms of the OMBM problem and clarify their strengths and weaknesses, which can guide practitioners to select appropriate algorithms for various scenarios.

1. INTRODUCTION

Given a set of service providers and a set of users in a 2D space, the minimum bipartite matching in spatial data (MBM) problem aims to find a maximum-cardinality matching with minimum total distance between the matched pairs and has attracted much attention from the database communities in the last decade [31, 28, 32]. With the unprecedented development of mobile Internet and smartphone techniques in recent years, many applications of the MBM problem on real-time spatial data become popular, such as the real-time taxi-calling service Uber [5], the on-wheel meal-ordering service GrubHub [2], and the product placement checking service of stores Gigwalk [1]. To deal with the minimum bipartite matching in real-time dynamic spatial environments, a natural way is to model it as the online minimum bipartite matching in real time spatial data (OMBM) problem [16, 17].

Though with the same objective function, the traditional MBM problem and the OMBM problem address different scenarios and constraints. Specifically, traditional MBM addresses the *offline scenario*, where full information of service providers and users is known before matching is conducted, while OMBM addresses the *online scenario*, where (1) each service provider has an initial location, but users dynamically arrive one by one; (2) before a user appears, his/her location is unknown; (3) once a user appears, s/he must be matched to one unmatched service provider immediately before subsequent users arrive. Particularly, there are a wide range of real applications of the OMBM problem, and several representative examples are shown as follows:

- **Task Assignment in Spatial Crowdsourcing [7]:** Task assignment is one of the most foundational issues in spatial crowdsourcing[11, 18, 28, 24, 25, 26, 27, 29, 30]. In real-time spatial crowdsourcing, a task can be considered as a user and a crowd worker can be considered as a service provider. The goal of task assignment in spatial crowdsourcing is usually to minimize the total travel distance of workers. In particular, in real applications, each task request not only dynamically appears but also needs to be assigned to a crowd worker as quickly as possible. Thus, task assignment in real-time spatial crowdsourcing can be addressed by the OMBM problem.
- **Taxi Dispatching [20, 23]:** Taxi dispatching systems are very popular in current daily life. One representative commercial application is Uber [5]. A taxi and a calling-taxi request can be considered as a service provider and a user, respectively. The taxi dispatching system usually tries to minimize the total waiting time of users or the total driving distance for the taxies to pick up their passengers. Note that each dynamically appearing calling-taxi request should be immediately responded once it appears. Therefore, OMBM is suitable for handling such real-time allocation in the taxi dispatching systems.
- **Wireless Network Connection Management [31]:** In wireless network connection management, WiFi receivers and wireless access points (APs) can be regarded as users and service providers in the OMBM problem, respectively, where each WiFi receiver is only allocated a nearby AP immediately once it requests for WiFi service, each AP can provide WiFi service for multiple WiFi receivers, and the overall distances between WiFi receivers and APs should be minimized

Proceedings of the VLDB Endowment, Vol. 9, No. 12

to provide high-quality network service. Although each single AP has a capacity, which is the maximum number of WiFi receivers it can support, the AP can be considered as multiple APs with capacity of one, each of which can support only one WiFi receiver. This multiple (WiFi receivers)-to-single (AP) assignment problem can be reduced to a one-to-one assignment problem. Therefore, OMBM can be naturally applied to handling such applications.

With its wide applications, the OMBM problem has been extensively studied, and some of the most notable algorithms include Greedy [16, 17], Permutation [16, 17, 19], HST-Greedy [22], and HST-Reassignment [8, 9]. However, study on the performance of the algorithms in practice is still absent. This paper is the first work to evaluate the performance of these algorithms through a comprehensive experimental study with additional theoretical analysis.

1.1 Motivation

1. Is Greedy really the worst? Greedy [16, 17] is the most simple and efficient solution for the OMBM problem. The basic idea of Greedy is to allocate each new arrival user to the currently nearest unmatched service provider. Because existing studies mainly focus on theoretically analyzing the worst-case competitive ratio of an online algorithm, which is the worst-case ratio of the total distance of the matching returned by the online algorithm to that of the optimal matching (which can be obtained in the offline scenario), Greedy has been considered as the worst algorithm due to its exponential worst-case competitive ratio. However, a comprehensive experimental comparison of the proposed algorithms for the OMBM problem is still absent so far. Therefore, whether Greedy is really ineffective in practice is unknown.

2. Is the worst-case analysis appropriate for the OMBM problem in practice? As discussed above, most existing studies use the worst-case competitive ratio to evaluate the effectiveness of their proposed algorithms. However, through extensive experiments on real and synthetic datasets, we observe some contradictions between the performance of the proposed algorithms and their theoretical results. For example, according to [8, 9, 14], both HST-Greedy [22] and HST-Reassignment [8, 9] have better worst-case competitive ratios than that of Greedy. However, according to our experiment results, Greedy is significantly superior to both HST-Greedy and HST-Reassignment with more than 50,000 tests on real and synthetic datasets. Therefore, these contradictions raise the question that whether the worst-case analysis is appropriate for the OMBM problem in practice.

3. Are implementations and experimental evaluation uniform? To avoid that different implementation details result in inconsistent performance evaluation of the algorithms, it is necessary to provide a fair experimental comparison for the existing algorithms and report their real contributions. For instance, comparing HST-Greedy and HST-Reassignment requires uniform implementation of the hierarchically separated tree (HST) structure [12]. In addition, since there is no previous experimental study for the OMBM problem, the selection of datasets and the experimental design are also important.

1.2 Contributions

1. Good performance of Greedy: We explore the performance of Greedy with more than 50,000 tests on real and synthetic datasets following four different representative location distributions. Surprisingly, we observe that the simple and efficient greedy algorithm, which has been considered as the worst in theoretical analysis, is actually more effective than other existing algorithms in almost all the tests. Furthermore, Greedy not only has outstanding scalability but also has comparative ratio as low as 5, and usually lower than 2, in all different cases. In particular, the strategy adopted by Greedy is equivalent to conducting a nearest neighbour query for each user, which has been widely studied and can be easily extended into many applications in the database community. Thus, the outstanding performance of Greedy also provides hints to other applications in the database community.

2. Worst-case vs. Average-case analysis: Inspired by the big gap between the experimental results and the existing theoretical analysis of Greedy, we discover that the worst case of Greedy rarely occurs in practice. Thus, we believe that the worst-case analysis may be not appropriate for the OMBM problem in real applications and the average-case analysis should be more suitable. We introduce the average-case analysis model of online algorithms, called the random order model, and revisit that the competitive ratio of the worst case of Greedy in the worst-case analysis is actually just a constant, 3.195, in the average-case analysis in Section 4.

3. Uniform implementations and experiments: We present efficient implementation for the four representative algorithms, including Greedy, Permutation, HST-Greedy, and HST-Reassignment. These implementations adopt common basic operations (e.g. the construction of the HST structure) and offer a base for comparison with future work in this area. Moreover, the source code and datasets used in the experiments are available in [4]. In addition to uniform implementations, we also study on a large real dataset, which consists of real-time taxi-calling data in more than half year, and five synthetic datasets, where locations of service providers and users are randomly generated following different commonly used distributions (i.e. normal distribution, uniform distribution, power law distribution and exponential distribution) to eliminate the bias of a particular dataset towards the algorithms.

4. Potential open questions: Although we still cannot prove that the competitive ratio of Greedy in the average-case analysis is a constant, the aforementioned extensive random experiment results motivate us to propose the following hypothesis as a open question: *the average-case competitive ratio under the random order model of Greedy for the OMBM problem should be constant*, which can provide a theoretical explanation for the outstanding performance of Greedy in practice if the hypothesis holds.

2. PRELIMINARIES

2.1 Problem Definition

We formally define the online minimum bipartite matching in real time spatial data (OMBM) problem as follows.

DEFINITION 1 (OMBM PROBLEM). *Given a set of service providers W with specific locations, a set of users T whose spatial information is unknown before they appear, and a metric distance function $dis(.,.)$ in 2D space, the OMBM problem is to find a matching M to minimize the total distance $Cost(M) = \sum_{t\in T, w\in W} dis(t, w)$ between the matched pairs such that the following constraints are satisfied:*

- *Real-time constraint: once a user appears, a service provider must be immediately allocated to her/him before the next user appears.*
- *Invariable constraint: once a service provider is allocated to a user, the allocation cannot be revoked.*
- *Cardinality constraint: $k = |M| = \min\{|T|, |W|\}$, where $|.|$ is the size of a given set.*

The OMBM problem is illustrated by the following example.

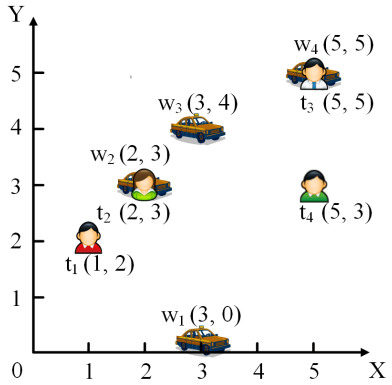


Figure 1: Locations of service providers (taxis) and users (tasks)

Table 1: Arrival order of four taxi-calling tasks

Arrival Order	1st	2nd	3rd	4th
1st Order	t_1	t_2	t_3	t_4
2nd Order	t_3	t_4	t_2	t_1

EXAMPLE 1. *Suppose a taxi dispatching platform has four service providers (taxis) ($w_1 - w_4$) and four taxi-calling task ($t_1 - t_4$) from four users. The locations of the taxis and users (revealed as they arrive) are labeled in a 2D space (X, Y) in Figure 1. The platform wants to minimize the overall travel distance cost, e.g. Euclidean distance, for the assigned taxis to pick up the users. The taxis are assumed to be relatively static in a time interval (e.g. 10 minutes) and their locations are known in advance, and the users dynamically appear.*

Table 1 shows two different arrival orders of the users. In the offline scenario, where the locations of users are known, the offline optimal matching is $(t_1, w_1), (t_2, w_2), (t_3, w_4), (t_4, w_3)$ with cost $2\sqrt{2} + \sqrt{5} \approx 5.06$. Notice that a taxi should be immediately allocated to each new-arriving user in the online scenarios. The simple greedy strategy, Greedy, is to allocate each new-arriving user to its currently nearest unmatched taxi. For the "2nd order", the matching returned by Greedy is the same as the offline optimal matching. However, for the "1st order", the cost of Greedy is 6.43, which is worse than that of the offline optimal matching. It indicates that the arrival orders of users usually affect the effectiveness of an online algorithm.

2.2 Competitive Analysis Models

In this subsection, we formally introduce the evaluation standard *competitive ratio (CR)* for online algorithms, which is the ratio of the result of an online algorithm to the optimal result, which can be obtained in the offline scenario. Since the arrival orders of objects significantly affect the performance of an online algorithm, different evaluation approaches of competitive ratios take different assumptions on the online arrival orders of the dynamically arrived objects. In the following, we introduce two representative competitive ratios under two kinds of online arrival order assumptions, the adversarial model (the worst-case analysis) and the random order model (the average-case analysis), for the OMBM problem.

DEFINITION 2 (CR IN THE ADVERSARIAL MODEL). *The competitive ratio of an online algorithm in the adversarial model for the OMBM problem is as follows:*

$$CR_A = max_{\forall G(T,W) \text{ and } \forall \sigma \text{ of } T} \frac{Cost(M)}{Cost(OPT)} \quad (1)$$

where $G(T, W)$ is an arbitrary metric bipartite graph of service providers and users, where the weight of an edge in the $G(T, W)$ corresponds to the distance between the two objects in T and W respectively, σ is an arbitrary arrival order of the users in the T, $Cost(M)$ is the total distance cost generated by the online algorithm, and $Cost(OPT)$ is the offline optimal total distance cost.

Note that the aforementioned $Cost(OPT)$ can be calculated by classical offline MBM algorithms, e.g. the successive shortest path algorithm (SSPA) [6] or the Hungarian algorithm [10] given full information of service providers and users in advance. In a word, the competitive ratio in the adversarial model is the worst-case analysis and always considers the worst-case ratio over all possible inputs and all possible arrival orders.

DEFINITION 3 (CR IN THE RANDOM ORDER MODEL). *The competitive ratio of an online algorithm in the random order model for the OMBM problem is as follows:*

$$CR_{RO} = max_{\forall G(T,W)} \frac{\mathbb{E}[Cost(M)]}{Cost(OPT)} \quad (2)$$

where $G(T, W)$ is the same as that in the adversarial model, $\mathbb{E}[Cost(M)]$ is the expectation of the total distance cost of the online algorithm over all possible arrival orders of T in the specific $G(T, W)$, and $Cost(OPT)$ is the offline optimal total distance cost.

The random order model adopts the average-case analysis and measures the worst average performance of an online algorithm. In other words, among all the average ratios of an online algorithm over all possible metric bipartite graphs, where each average ratio is the expected performance of the algorithm over all possible arrival orders for a specific graph instance, the random order model focuses on the worst average one. On the contrary, the competitive ratio under the adversarial model is to bound the worst-case performance of an online algorithm over all possible cases, i.e. arrival orders. All existing studies for the OMBM problem focus on the adversarial model but ignore the average performance of the algorithms. However, as discussed later, we discover that the competitive ratio analysis under the random order model may be more suitable for evaluating the performance of the online algorithms for the OMBM problem in practice because the special worst cases, which will be introduced in Section 4, rarely occur in real applications.

3. ONLINE ALGORITHMS

In this section, we describe the main ideas of each online algorithm compared by our experimental study. We categorize the four online algorithms into two groups, deterministic algorithms and randomized algorithms, respectively.

3.1 Deterministic Algorithms

3.1.1 Greedy Algorithm

We first introduce the online greedy algorithm, Greedy, which was presented by [16]. The main idea of Greedy is to match each new arrival user to its currently nearest unmatched service provider. For example, based on our running example in Example 1, the result of Greedy is $(t_1, w_1), (t_2, w_2), (t_3, w_4), (t_4, w_3)$ if users appear following the "1st order". Although Greedy is very efficient, its competitive ratio in the adversarial model is proven to be $2^k - 1$, where $k = |M| = \min\{|T|, |W|\}$ is the maximum cardinality of the matching. Hence, Greedy is always considered as the worst solution for the OMBM problem. The worst case of Greedy is further studied in Section 4.

Algorithm 1: 2-HST Construction Algorithm

Input: metric $\{V, d\}$
Output: an HST metric $\{V', d_T\}$

```
1  Choose a random permutation π of V;
2  Choose β in [1,2] randomly from the distribution
   p(x) = 1/(x ln 2);
3  Δ ← max_{u,v∈V} d(u, v);
4  δ ← ⌈log_2 Δ⌉;
5  D_δ ← {V};
6  i ← δ − 1;
7  while D_{i+1} is not a singleton cluster do
8      β_i ← 2^{i−1} β;
9      for l ← 1, 2...k do
10         foreach clusters S in D_{i+1} do
11             Create a new cluster consisting of all unassigned
               vertices in S closer than β_i to π(l);
12             Mark the vertices in the new cluster assigned;
13             Join the new cluster with S by edge of length
               2^{i+1};
14     i ← i − 1;
15 return an HST
```

3.1.2 Permutation Algorithm

Let T_i denote the set of users arriving before the i-th user t_i arrives. The Permutation algorithm mainly includes the following four steps. (1) After t_i appears, Permutation conducts the classical offline minimum weighted matching algorithm, e.g. Hungarian algorithm, on the bipartite graph $G(T_i, W)$ and gets a minimal weighted partial matching [16], denoted by M_i. (2) If the service provider w_i matched to t_i in M_i is unmatched in the online matching result, the pair (t_i, w_i) is matched in the final online matching result. (3) Otherwise, it is guaranteed that there exists exactly one service provider w_j that does not appear in M_{i-1}, and Permutation matches w_j to t_i, namely adding (t_i, w_j) to the final result. The algorithm is named as Permutation due to its aforementioned permutation property. Since the upper bound on the cost of the permutation in this algorithm can be proven to be $2i - 1$ when the i-th user appears, the competitive ratio of Permutation is $2k - 1$ with $i = k = |M| = \min\{|T|, |W|\}$. To further illustrate the Permutation algorithm, we go through the following example.

EXAMPLE 2. *Taking our running example in Example 1, when the first user t_1 appears, Permutation gets its minimal weight partial matching (t_1, w_2). Then when t_2 arrives, the minimal weighted partial matching is $(t_1, w_1), (t_2, w_2)$. But t_1 is already matched to w_2 which cannot be revoked, and thus t_2 is matched to the currently unmatched service provider w_1 in M_2. Similarly, t_3 and t_4 are allocated to w_4 and w_3, respectively. The final matching result is $(t_1, w_2), (t_2, w_1), (t_3, w_4), (t_4, w_3)$ with cost* 6.81.

3.2 Randomized Algorithms

In this subsection, we mainly introduce two randomized online algorithms, HST-Greedy and HST-Reassignment, for the OMBM problem. Since both algorithms utilize a structure, called hierarchically separated tree (HST), we first introduce the HST structure and then review the two algorithms.

3.2.1 Hierarchically Separated Tree (HST)

Since the HST structure can only be applied to a metric space, we first introduce the concept of metric space. A ***metric space*** is denoted as a pair (V, d) where V is a set of objects and $d : V \times V \to [0, \infty)$ is a *metric*, satisfying the following three axioms: (1) $d(u, v) = 0$ if and only if $u = v$ $(u, v \in V)$, (2) $d(u, v) = d(v, u)$, and (3) $d(u, v) + d(v, w) \leq d(u, w)$, i.e. the triangle inequality. For example, a 2D space $\mathbb{R}^2$ with Euclidean distance d is a metric space. An arbitrary given metric space can be projected to a *hierarchically separated tree (HST)* metric space, which not only has several sound properties but also provides theoretical bound on the distortion between the two metric spaces. The HST is defined as follows.

α-Hierarchically Separated Tree (α-HST). Given a metric (V, d), we say the HST metric (V', d_T) approximates the original metric in two ways. First, it needs to dominate the original metric (V, d). Here "dominate" means that for all u,$v \in V$, $d_T(u, v) > d(u, v)$. Also, it guarantees $\mathbb{E}[d_T(u, v)] \leq \mathcal{O}(\alpha \log |V|)d(u, v)$. Let $d_T(.,.)$ be the length of the unique shortest path between two vertices. In other words, given two arbitrary vertices in the HST, the distance between them, $d_T(u, v)$, is the sum of the distances along the shortest paths from u, v to their lowest common ancestor in the HST. Then, the HST has the following four properties on the distance metric[12]:

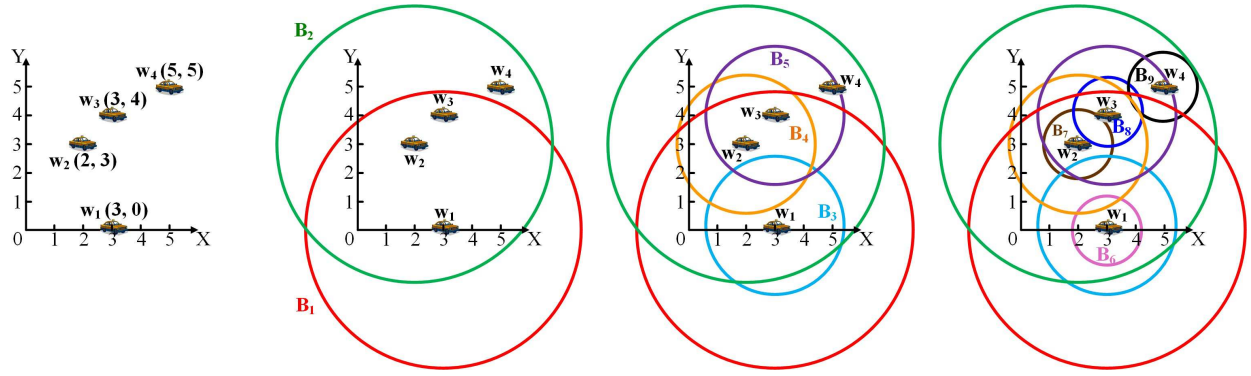
- It is a rooted tree. The root vertex contains the whole set V, and each leaf vertex corresponds to an unique object in the set V. Each of the other vertices contains a subset of V, which is the union of the sets of objects contained in its children.
- For an arbitrary vertex $s \in V'$, if $c_1(s)$ and $c_2(s)$ are the children of s in the HST, $d_T(s, c_1(s)) = d_T(s, c_2(s))$.
- For an arbitrary vertex $s \in V'$, let $p(s)$ be the parent of s and $c(s)$ be a child of s, then $d_T(s, p(s)) = \alpha d_T(s, c(s))$.
- All the leaf vertices are at the same level of the HST. For an arbitrary vertex $s \in V'$, let $\lambda_1(s)$ and $\lambda_2(s)$ be the leaf vertices that are the descendants of s, then $d_T(s, \lambda_1(s)) = d_T(s, \lambda_2(s))$.

Note that the α-HST provides theoretical guarantee regarding the expected value of the distance $\mathbb{E}[d_T(u, v)]$ for two arbitrary given vertices in the HST. The bound is for the expected value because the HST construction algorithm is a randomized algorithm, more details of which will be introduced later. Furthermore, the parameter α of an α-HST is the unit distance and is usually set as 2 in practice. In the remaining parts of this paper, we set $\alpha = 2$ and use 2-HST as an example to illustrate the concept of HST.

In general, HST is usually used as a tool to approximate some metrics. e.g. Euclidean metric. When we transform the problem from the original metric into a tree metric, we can utilize the sound properties of the tree metric, such as recursiveness and symmetry. Thus, efficient online algorithms can be designed and implemented. In the following, we will introduce the 2-HST construction algorithm.

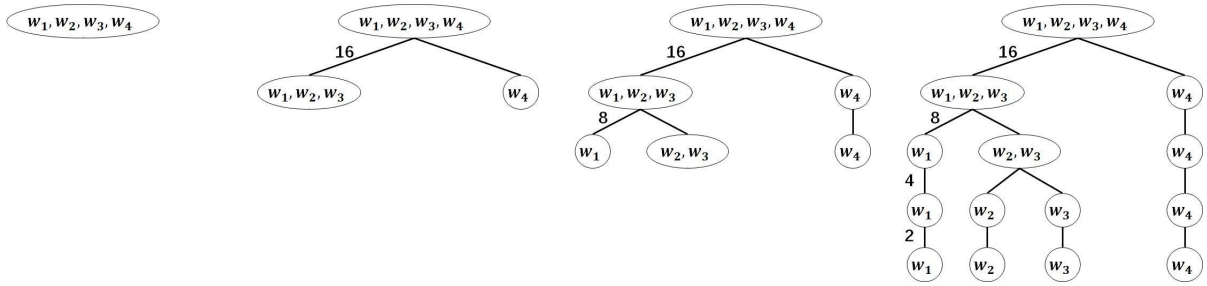
The main idea of the 2-HST construction algorithm is to first randomly generate a global permutation of all the given objects as an order, and then performs a hierarchical decomposition following the randomly generated order level by level. Finally, the hierarchical decomposition of the original set of objects results in a rooted tree as follows. Each vertex in the tree contains a decomposed set of objects while the root contains the whole set V, and the leaves are singletons. Particularly, the distance between a pair of parent-child vertices in the (i+1)-th and the i-th levels respectively is exactly $2^{(i+1)}$.

The procedure of the 2-HST construction process is illustrated in Algorithm 1. In lines 1-2, the algorithm randomly generates a



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4

Figure 2: The Open Balls in Each Decomposition Step.



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4

Figure 3: A 2-HST Construction Process based on The Open Balls.

global permutation order for all the objects and a random parameter β. In lines 3-6, Δ is set as the diameter of all the objects in the original metric space, the height of the HST is $\delta = \lceil \log_2 \Delta \rceil$, and the root vertex D_δ in the 2-HST contains the whole set of objects. Lines 7-14 perform a top-down hierarchical decomposition. For the (i+1)-th level in the 2-HST, as long as a vertex in this level contains more than one object, i.e. a non-singleton vertex, the algorithm iteratively processes each object according to the random global permutation order and finds the objects locating in the ***open ball*** centered at the location of the currently iterated object with the radius of β_i. Such objects are grouped to generate a new vertex in the i-th level, whose parent is the original vertex in the (i+1)-th level. More specifically, the object located in the open ball of radius β_i centered at the location of object u is defined as a set of objects such that $b(u, \beta_i) = \{v \in V | d(u, v) < \beta_i\}$. The whole algorithm terminates until all vertices are singleton.

As mentioned above, as the HST construction algorithm (Algorithm 1) is a randomized algorithm, it can only provide theoretical guarantee for $\mathbb{E}[d_T(u, v)]$. Specifically, there are two reasons. On one hand, the HST construction algorithm first generates a random permutation of all the objects (all service providers in our paper) for the remaining partitions. Even though for the same set of objects, the HST construction algorithm may build different HST structures due to different random permutations of the objects. On the other hand, for the partition in the i-th level in a 2-HST, the radius of the open ball is $\beta_i = 2^{i-1}\beta$, where β is a global parameter generated randomly from the interval [1, 2] with distribution $p(x) = \frac{1}{x \ln 2}$. Since both the permutation order of all the objects and the parameter to calculate the radius of open balls are generated randomly, HST can only provide theoretical guarantee for the expected value of $d_T(u, v)$. We further illustrate the HST construction algorithm by the following example.

EXAMPLE 3. *Back to our running example in Example 1, V is the set of four service providers (taxis) in the 2D metric space shown in Figure 2a. Suppose we choose $\beta = 1.2$ and the global permutation is $< w_1, w_2, w_3, w_4 >$. $\Delta = \max_{w_i, w_j \in W} d(w_i, w_j) = \sqrt{29}$ and $\delta = \lceil \log \sqrt{29} \rceil = 4$. The root vertex in the 2-HST contains four taxis, which is shown in Figure 3a. Then, the algorithm partitions the root vertex into disjoint subsets of objects in level 3, where $\beta_3 = 2^{3-1} \times 1.2 = 4.8$. Based on the global permutation, the algorithm first finds the open ball with radius 4.8 and the center of w_1, which is shown as the red circle B_1 in Figure 2b. Similarly, the open balls of w_1, w_2, w_3 are empty, empty and $\{w_4\}$, respectively. Thus, we only show the open ball of w_4 as the green circle B_2 in Figure 2b. With open balls $B_1 = b(t_1, \beta_3) = \{w_1, w_2, w_3\}$ and $B_2 = b(t_4, \beta_3) = \{w_4\}$, the HST decomposes the root vertex into two vertices in the 3rd level in the HST, which are shown in Figure 3b. Similarly, for the 2nd level, the radius of the open balls is $\beta_2 = 2.4$, and there are three open balls, $B_3 = w_1$ (the blue circle), $B_4 = w_2, w_3$ (the yellow circle), and $B_3 = w_4$ (the purple circle), in Figure 2c. And the corresponding decomposition in the 2nd level of the 2-HST is shown in Figure 3c. In the 1st level, the radius of the open balls is $\beta_1 = 1.2$, and there are four open balls, each of which is singleton as shown in Figure 2d. And the corresponding decomposition in the 1st level of the 2-HST is shown in Figure 3d. The algorithm terminates in level 0. As HST requires that every two vertices are at least 1 unit away and the vertices in level 0 have radius at most 1.*

3.2.2 HST-Greedy Algorithm

HST-Greedy [22] first builds an α-HST structure for the service providers, where all the service providers are projected onto a tree

Table 2: Four evaluated algorithms of the OMBM problem in this paper

Algorithms	Input Data	Time Complexity per Each Arrival Vertex	Randomization	Data Structure	Competitive Ratio (Adversary Model)
Greedy [16]	Metric space data	$\mathcal{O}(k)$	Deterministic	-	$\mathcal{O}(2^k - 1)$
Permutation [16]	Metric space data	$\mathcal{O}(k^3)$	Deterministic	-	$\mathcal{O}(2k - 1)$
HST-Greedy [22]	Metric space data	$\mathcal{O}(k)$	Randomized	HST	$\mathcal{O}(log^3 k)$
HST-Reassignment [9]	Metric space data	$\mathcal{O}(k^2)$	Randomized	HST	$\mathcal{O}(log^2 k)$

metric. HST-Greedy includes the following two main steps to process each new arrival user t_i: (1) HST-Greedy first finds the service provider v_i currently nearest to t_i in the original 2D space. (2) HST-Greedy then chooses an unmatched service provider w_i nearest to v_i on the tree metric. If there are multiple service providers that have the same distance to v_i on the tree metric, the algorithm randomly chooses one as w_i. If v_i is also an unmatched service provider, w_i is replaced by v_i to be matched to t_i. Otherwise, w_i is directly matched to t_i. Thus, the pair (t_i, w_i) is added to the final online matching. With the α-HST structure, the total cost of HST-Greedy on the tree metric is $\mathcal{O}(\log k)$ when $\alpha > 2 \ln k + 1$. In addition, α-HST can also guarantee that the expectation of the distance of two vertices on the tree is no greater than $\alpha \log k$ times the original distance. Therefore, the final competitive ratio of HST-Greedy is $\mathcal{O}(\log^3 k)$.

3.2.3 HST-Reassignment Algorithm

Different from HST-Greedy that adopts an α-HST structure ($\alpha > 2 \ln k+1$), HST-Reassignment [9] only uses 2-HST structure, namely $\alpha = 2$. The main idea of HST-Reassignment is similar to HST-Greedy. When a new user t appears, HST-Reassignment also first finds the service provider v_i currently nearest to t_i in the original 2D space. The main difference between HST-Greedy and HST-Reassignment is their second step. HST-Greedy directly finds the nearest unmatched service provider w_i for v_i, but it is likely that HST-Greedy is trapped into the local optimal solution such that the total distance cost of the final matching is very expensive. To avoid the local optimal traps, HST-Reassignment designs a reassignment approach, whose basic idea is to iteratively change w_i from the previously matched pairs until it finds an unmatched service provider who is a sight farther unmatched service provider of v_i on the tree metric. In the competitive analysis, a Restricted Reassignment Model is proposed that guarantees HST-Reassignment to have competitive ratio of $\mathcal{O}(\log^2 k)$ in the adversarial model. Note that even though HST-Reassignment obtains a better competitive ratio, its effectiveness is worse than that of HST-Greedy and Greedy in practice according to our experiments. More experimental results will be discussed in Section 5.

3.2.4 Summary

Table 2 summarizes all the aforementioned algorithms that we review and evaluate in this paper.

4. GREEDY REVISITED

[16] indicates that the worst case of Greedy under the adversarial model is when all the vertices lie on a line. In this section, we inspect the properties of such worst case of the adversarial model under the random order model. Particularly, we show that the competitive ratio of this worst case under the random order model is 3.195, which is a constant. We next review this "bad" example and analyze that the worst case w.r.t. Greedy in this example only appears with very low probability of $\frac{1}{k!}$, where $k = \min\{|T|, |W|\}$.

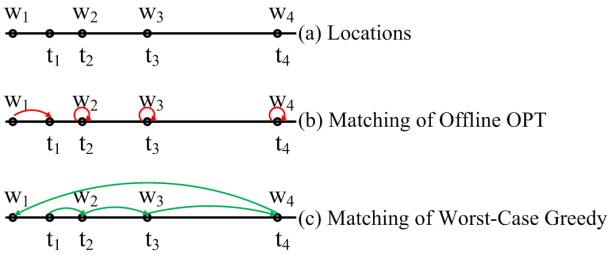


Figure 4: Offline OPT v.s. Worst-case of Greedy

A "Bad" Example [14, 16]. Consider k service providers, $w_1 - w_k$, located at points $\{-\epsilon, 2, 2^2, \cdots, 2^{k-1}\}$ on a line respectively, where all the coordinates are integers and ϵ is an arbitrarily small positive number. Moreover, k users, $t_1 - t_n$, appear at points $\{1, 2, 2^2, 2^3, \cdots, 2^{k-1}\}$ on the same line respectively. Figure 4 shows the "bad" example instance, which consists of four service providers and four users. Figure 4(a) shows the locations of the service providers at points $\{-\epsilon, 2, 4, 8\}$ and those of the users at points $\{1, 2, 4, 8\}$ on the line, respectively. The matching result of offline OPT is shown in Figure 4(b), and its cost, the total distance, is $1 + \epsilon + 0 + 0 + 0 = 1 + \epsilon$. As for Greedy, the worst-case arrival order of the users in the bad example is $< t_1, t_2, \cdots, t_n >$, which results in the matching with cost $2^k - 1$. Figure 4(c) shows the matching result of Greedy for the worst-case arrival order $< t_1, t_2, t_3, t_4 >$, which has cost of $1 + 2 + 4 + 8 = 16$.

LEMMA 1. *Given the aforementioned "bad" example, where k service providers and k users lie on a line with integer coordinates, the worst-case matching result of Greedy only appears with probability* $\frac{1}{k!}$.

PROOF. In the "bad" example, each user t_i ($i \geq 2$) has a nearest service provider at the same location with it except t_1. If such nearest service provider for t_i ($i \geq 2$) is available (unmatched when t_i arrives), the cost between such pair is zero. Hence, for an arbitrary arrival order of users, the cost of its corresponding matching is lower than that of the worst-case matching as long as at least one user t_i ($i \geq 2$) arrives before t_1 arrives and thus the corresponding zero-cost service provider of t_i ($i \geq 2$) will not be occupied before t_i arrives. In other words, only the arrival order of $< t_1, t_2, \cdots, t_n >$ results in the worst matching cost. Therefore, the worst case only appears with the probability of $\frac{1}{k!}$. □

THEOREM 1. *Given the aforementioned "bad" example with k service providers and k users lying on a line with integer coordinates, the competitive ratio of Greedy under the random order model is 3.195.*

PROOF. According to the definition of the "bad" example, Greedy always assigns the nearest service provider available to a new-arrival user. Thus, for each user t_i ($i \in \{1, \cdots, k\}$), its cost is only one of the following two possible values,

$$Cost(t_i) = \begin{cases} 0 & \text{with probability } 1 - \frac{1}{i!} \\ 2^{i-1} & \text{with probability } \frac{1}{i!} \end{cases} \quad (3)$$

Furthermore, for each user t_i, its cost is 2^{i-1} if and only if all the users $t_j (j < i)$ appear before t_i arrive. Otherwise, its cost is zero. Therefore, the expected cost of each user t_i is

$$\mathbb{E}[Cost(t_i)] = \frac{2^{i-1}}{i!} \tag{4}$$

Since there are k users, the expectation of the total distance is

$$\mathbb{E}[Cost(M)] = \sum_{i=1}^{k} \frac{2^{i-1}}{i!} \tag{5}$$

Based on Equation (5), we prove that the expectation of the total distance is 3.195 as follows.

Since the series $\sum_{i=1}^{\infty} \frac{2^{i-1}}{i!}$ must be an upper bound of the expectation of the total distance, we analyze the bound of this series. We define the remainder term of the series $\sum_{i=1}^{\infty} \frac{2^{i-1}}{i!}$ as $R_N = \sum_{i \leq N+1}^{\infty} \frac{2^{i-1}}{i!}$. Based on the inequation $n! > (\frac{n}{e})^n$ $(n = 1, 2, \cdots)$, when $i \leq N+1$

$$\begin{aligned} \frac{2^{i-1}}{i!} &< 2^{i-1} \left(\frac{e}{i}\right)^k = \frac{1}{i} \left(\frac{2e}{i}\right)^{i-1} \\ &= \frac{1}{N+1} \left(\frac{2e}{N+1}\right)^{i-1} = \frac{1}{2e} \left(\frac{2e}{N+1}\right)^{N+1} \left(\frac{2e}{N+1}\right)^{i-(N+1)} \end{aligned} \tag{6}$$

Thus, the remainder term R_N can be bounded

$$\begin{aligned} R_N &< \frac{1}{2e} \left(\frac{2e}{N+1}\right)^{N+1} \sum_{i=N+1}^{\infty} \left(\frac{2e}{N+1}\right)^{i-(N+1)} \\ &= \frac{1}{2e} \left(\frac{2e}{N+1}\right)^{N+1} \sum_{l=0}^{\infty} \left(\frac{2e}{N+1}\right)^{l} \end{aligned} \tag{7}$$

when $N \leq 4$, $\frac{2e}{N+1} < 1$. Hence, we have the upper bound of the remainder term R_N, $R_N < \frac{1}{2e} \left(\frac{2e}{N+1}\right)^{N+1} \frac{1}{1-\frac{2e}{N+1}}$. Let $N = 12$, we have

$$R_{12} = \frac{1}{2e} \left(\frac{2e}{13}\right)^{13} \frac{1}{1-\frac{2e}{13}} \leq \frac{13}{15.126e} \left(\frac{2e}{13}\right) 13 < 10^{-5} \tag{8}$$

Since we know $\sum_{i=1}^{11} \frac{2^{i-1}}{i!} < 3.19453$ and $R_{12} < 10^{-5}$, the series $\sum_{i=1}^{\infty} \frac{2^{i-1}}{i!} < 3.195$. Therefore, the expectation of the total distance $\sum_{i=1}^{n} \frac{2^{i-1}}{i!} < 3.195$ as well. □

To sum up, although the worst-case competitive ratio of Greedy is exponential, the worst matching cost appears with an extremely low probability, $\frac{1}{k!}$. In particular, for the "bad" example, we prove that the competitive ratio of Greedy under the random order model is 3.195. In other words, the average performance of Greedy is quite good in the "bad" example, which also motivates us to guess that the competitive ratio of Greedy on the OMBM problem under the random order model is a constant.

5. EXPERIMENTAL STUDY

In this section, we study the performance of four representative algorithms for the OMBM problem in practice. Particularly, we aim to provide uniform implementations for the algorithms and compare the real-world performance of the algorithms in a comprehensive way. Also, as the extensive experiment results indicate, we verify that the average performance of Greedy is not bad and it is very likely to have constant competitive ratio under the random order model.

Table 3: Synthetic dataset

Factor	Setting
μ_W^L (Mean of locations of service providers following normal distribution)	50, 75, **100**, 125, 150
σ_W^L (Variance of locations of service providers) following normal distribution)	5, 10, **15**, 20, 25
α_W^L (Shape of locations of service providers) following power-law distribution)	2, 2.5, **3**, 3.5, 4
λ_W^L (Scale of locations of service providers) following exponential distribution)	0.5, 0.75, **1**, 1.25, 1.5
Scalability	$\|T\| = \|W\|$ = 10K - 100K

5.1 Experiment Setup

Datasets. We first introduce the real and synthetic datasets.

Real Dataset. We use the taxi-calling data on the ShenZhou real-time taxi-calling platform [3] in four weeks in May 2015 in Beijing as the real dataset. Particularly, there were on average 15082 taxi-calling requests, which corresponds to a set of users, and 1263 private taxies, which corresponds to a set of service providers, each day. Notice that once a taxi was assigned to a task, both the taxi and the task would disappear from the platform and thus when the taxi finished its task and re-appeared on the platform, it can be taken as a new taxi instance/worker. Since each taxi serviced 10-15 tasks each day, there were on average 15364 workers each day in the dataset, which indicates that there were more workers than tasks. In Figure 5, we plot the average number of taxi-calling tasks in each five-minute time interval in a day. It shows that the tasks appear dynamically, and the numbers of tasks are particularly large in rushing hours around 8AM, 12PM, 6PM, and 10PM, respectively, indicating that it is necessary to apply online assignment algorithms in order to respond to the task requests in real-time. In addition, we randomly choose one day's data and present the location distribution of the task requests (users) and taxi instance/worker (service providers) in Figure 6. We observe that most tasks (blue markers) and workers (yellow markers) appeared in the central area of Beijing and only a small part of them appeared in the suburban district.

Synthetic Datasets. We generate 5000 users and 5000 service providers on a 200×200 2D grid, and randomly generate the locations of users and service providers following the commonly used Uniform and Normal distributions [13], and also Power Law and Exponential distributions. The similar approach of randomly generating test instances was used in previous artificial intelligence research [33]. Notice that Power Law and Exponential distributions are used since recent studies [21, 15] show that the movement of people and taxies usually follow these two distributions in cities. The statistics and configuration of synthetic data are illustrated in TABLE 3, where we mark our default settings in bold font. Notice that for the scalability test, we generate users and service providers on a 500×500 2D grid so that the 100K users and service providers will not overlap too much in location.

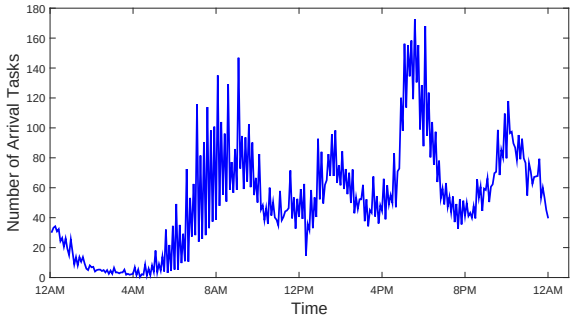


Figure 5: Average number of tasks of taxi-calling per day

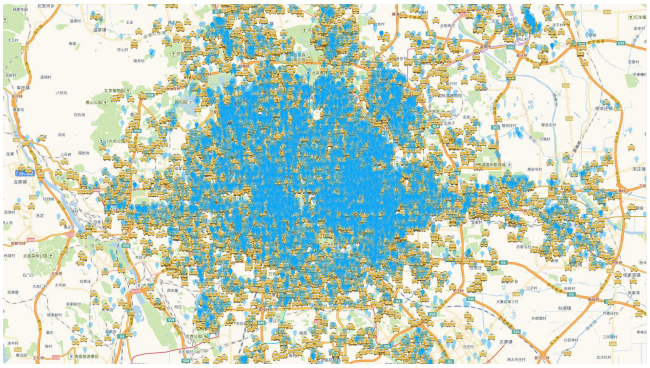


Figure 6: Location distribution of the users and service providers at the ShenZhou taxi-calling platform on one day in Beijing

Compared Algorithms and Experiment Environments. We study the performance of the algorithms in 2D space. Particularly, we compare the state-of-art online algorithms in 2D space, Greedy, Permutation, HST-Greedy and HST-Reassignment. We study the effect of varying parameters on the performance of the algorithms in terms of total distance, running time and memory cost. In particular, since the Permutation algorithm is very inefficient, we separately compare Permutation and Greedy in a small synthetic dataset. In each experiment, we repeatedly test 1000 different online arrival orders of users and report the average results. The algorithms are implemented in Visual C++ 2010, and the experiments were performed on a machine with Intel(R) Core(TM) i5 2.40GHz CPU and 4GB main memory.

5.2 Experiment Results

Effect of Locations of users following Normal distribution. Figure 7 shows the results when the locations of users follow Normal distribution and the locations of service providers follow three different distributions, respectively.

For total distance results, we can observe that Greedy is always better than HST-Greedy and HST-Reassignment and is nearly as good as the offline optimal algorithm. Particularly, Greedy is almost 2 times better than HST-Greedy and HST-Reassignment when the locations of service providers also follow the Normal distribution (Figures 7a and 7b), where the users and service providers are more concentrated and overlap more in locations. Notice that though the users and service providers are less concentrated as the standard deviation become larger (Figure 7b), Greedy is still much better. HST-Greedy is the runner-up. Also, the total distance of all the algorithms increases as σ_W^L increases in overall, which is because the average distance between a user and a service provider becomes larger as the locations of the service providers become less concentrated. However, the gap between the algorithms becomes narrower when the locations of service providers follow the Power Law and Exponential distributions (Figures 7c and 7d). The reason is that users and service providers have very small overlap in locations and a user is relatively far away from a service provider in these two cases, and thus the total distance generated by an arbitrary algorithm is mainly dominated by the distance between the set of users and the set of service providers.

The results of time and memory consumptions are presented in the last two rows in Figure 7. We can observe that Greedy is always more efficient in both time and space than the other two online algorithms since it only takes $O(|W|)$ time to process each user and does not need any extra space for storage of HST as the other two do. Since HST-Reassignment takes $O(|W|^2)$ time to process each user, it is the least inefficient algorithm among the three.

Effect of Locations of users following Exponential distribution. Figure 8 shows the results when the locations of users follow Exponential distribution and the locations of service providers follow three different distributions, respectively.

For total distance, we can again see that Greedy performs the best while HST-Greedy is better than HST-Reassignment for most of the time and Greedy is again nearly as good as the offline optimal algorithm. We can observe that all the algorithms have similar performance when the locations of service providers follow Normal distribution (Figures 8a and 8b). The reason is similar to that of Figure 7c and 7d where the set of users and the set of service providers do not overlap too much and are far away from each other and thus the total distance generated by an arbitrary algorithm is mainly dominated by the distance between the two sets. However, when users and service providers are mixed and overlapped in a concentrated area, i.e. locations of both sets follow similar distributions, Greedy performs much better than the two HST-based online algorithms (Figure 8c and 8d). Since in real applications, users and service providers usually overlap in locations and cannot be separated into two disjoint sets, the results indicate that Greedy can outperform other online algorithms. As for time and memory results, which are shown in the last two rows in Figure 8, we can again observe that Greedy is the most efficient in both time and space while HST-Reassignment is the most inefficient.

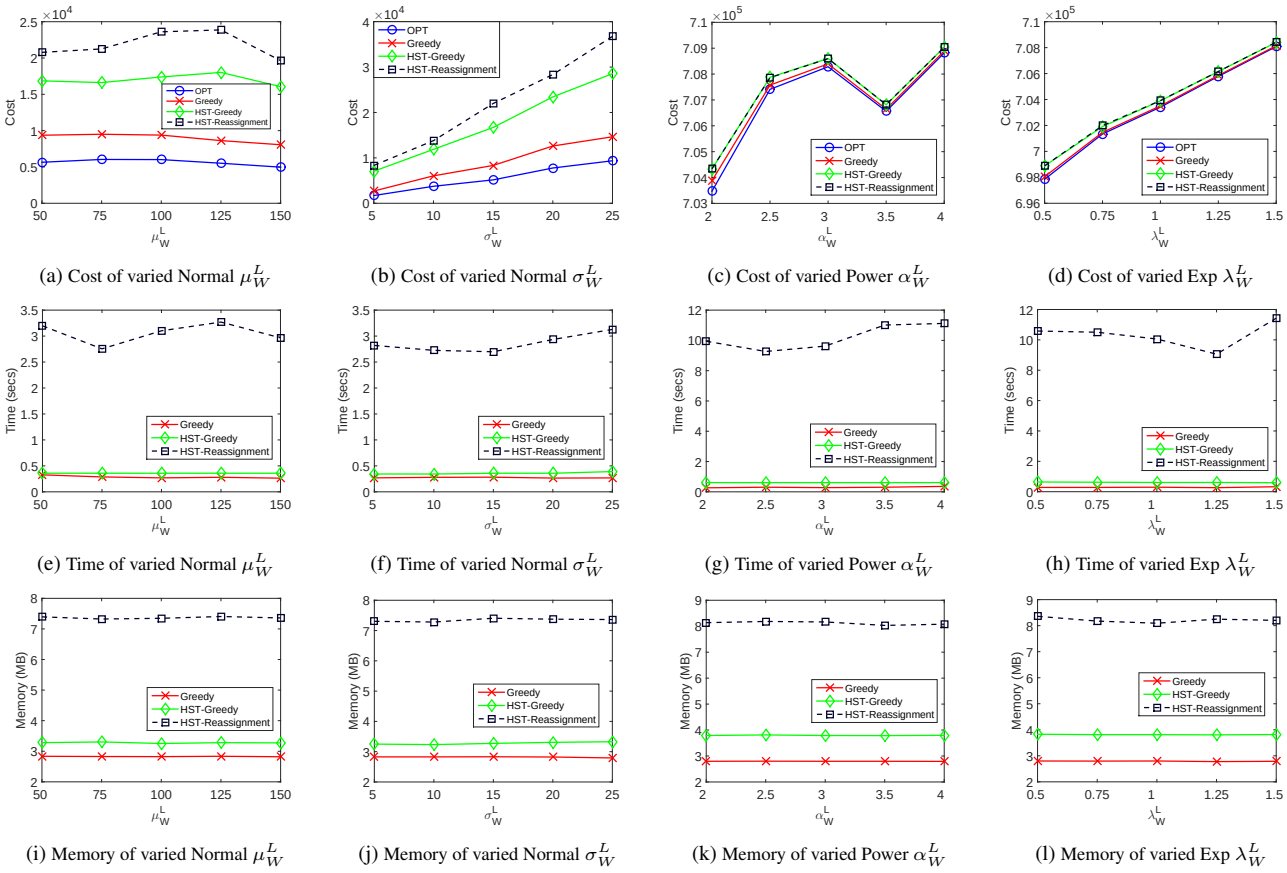
Effect of Locations of users following Uniform distribution. The total distance results when the locations of users follow Uniform distribution and the locations of service providers follow three different distributions, respectively, are presented in Figure 9. Since the time and space results are similar to the results in Figures 7 and 8, we omit them here for brevity.

We can again observe that Greedy performs the best in overall. Particularly, when the locations of service providers follow Normal distribution and we vary the mean of the distribution (Figure 9a), we can observe that the total distance of all the algorithms is quite low when the mean is at the center of the grid, i.e. point (100, 100), but is much larger when the mean is far away from the center of the grid. The reason is that the average distance between a user and a service provider becomes lower when the mean of Normal distribution is at the center of the grid and thus the service providers are concentrated around the center as the users are uniformly distributed across the grid. And when the mean of the Normal distribution is far away from the center of the grid, users are more far away from the service providers on average and thus the total distance is large. When the standard deviation of Normal distribution increases, the total distance decreases for all the algorithms (Figure 9b) because the locations of service providers are less concentrated.

Effect of Locations of users following Power-law distribution. Figure 10 shows the results when the locations of users follow Power-law distribution and the locations of service providers follow three different distributions, respectively. Again, we omit the time and space results as they are similar to previous results.

Again, we can see that Greedy generates lower total distance than the other two online algorithms in general. Also, similar to the previous results, all the algorithms have similar performance when the locations of users and service providers are distributed differently as Figures 10a and 10b show. However, the advantage of Greedy is more obvious when the locations of users and service providers are overlapped as Figures 10c and 10d show.

Scalability. We study the scalability of the algorithms in the first three columns of Figure 11, where the size of T/W is varied and the locations of users and service providers are generated following three different distributions, respectively. Notice that in our experiments, we terminate an algorithm if its running time is over



(a) Cost of varied Normal μ_W^L (b) Cost of varied Normal σ_W^L (c) Cost of varied Power α_W^L (d) Cost of varied Exp λ_W^L

(e) Time of varied Normal μ_W^L (f) Time of varied Normal σ_W^L (g) Time of varied Power α_W^L (h) Time of varied Exp λ_W^L

(i) Memory of varied Normal μ_W^L (j) Memory of varied Normal σ_W^L (k) Memory of varied Power α_W^L (l) Memory of varied Exp λ_W^L

Figure 7: Results that the locations of service providers in W follow Normal, Power-law, and Exponential distributions while the locations of users in T follow Normal distribution.

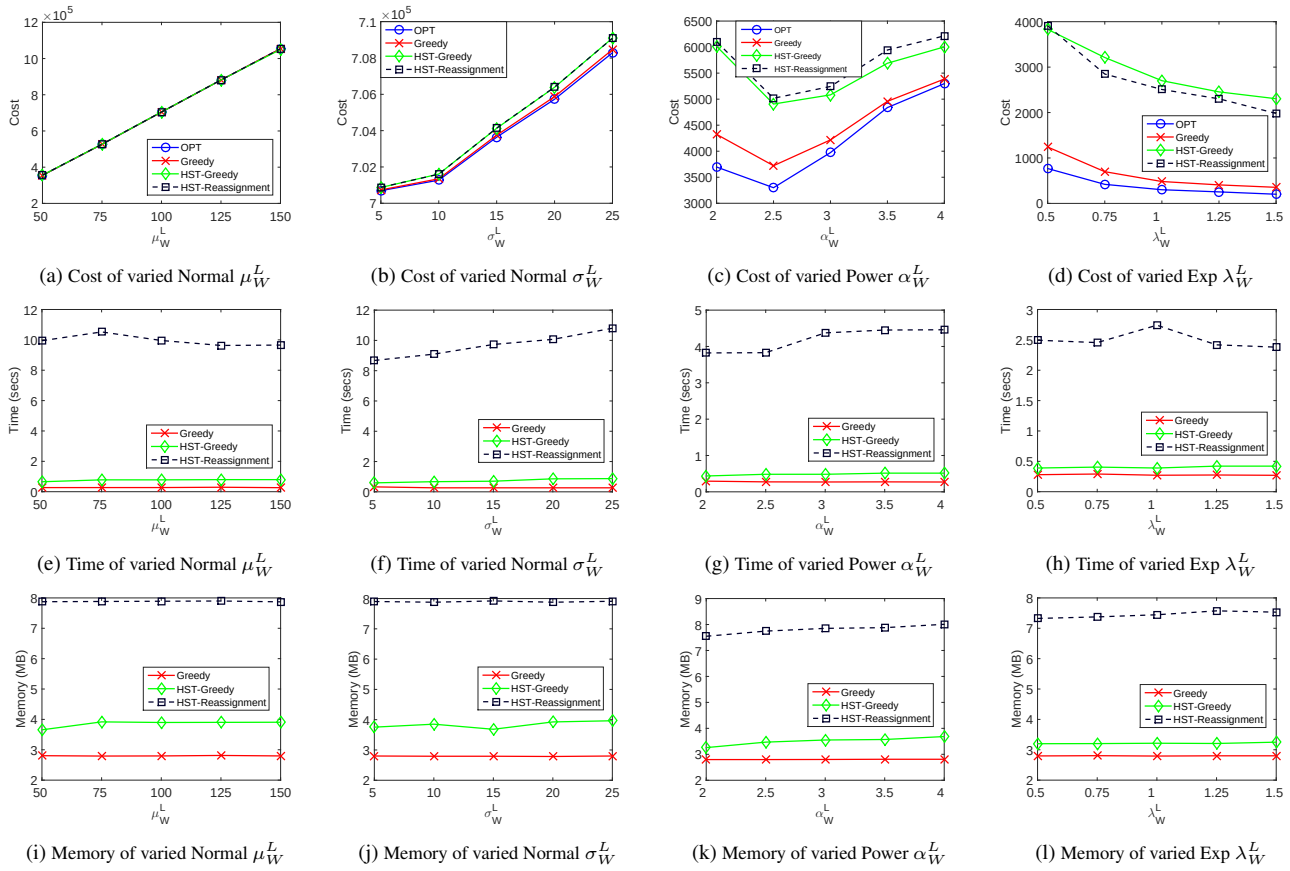
Table 4: Comparison of Permutation, Greedy, HST-Greedy and OPT

	Cost			Time(seconds)			Memory(MB)		
	T (Normal) W (Normal)	T (Uniform) W (Uniform)	T (Exp.) W (Power)	T (Normal) W (Normal)	T (Uniform) W (Uniform)	T (Exp.) W (Power)	T (Normal) W (Normal)	T (Uniform) W (Uniform)	T (Exp.) W (Power)
OPT	2066.38	6222.60	903.84	0.25	0.22	1.89	6.79	6.80	6.79
Greedy	3144.86	9607.34	936.51	0.03	0.03	0.03	2.60	2.61	2.61
Permutation	**3846.52**	**13104.90**	**1041.31**	**194.18**	**203.81**	**692.77**	**7.043**	**7.047**	**7.03**
HST-Greedy	5161.48	15883.1	1118.43	0.051	0.05	0.06	2.73	2.83	2.71
HST-Reassignment	7906.96	21120.00	1218.39	0.17	0.22	0.15	6.76	6.76	6.71

1800 seconds, and thus the results of the offline optimal algorithm are only available when $|T|(|W|) = 10K$ and 20K. For total distance, we can observe that Greedy is again the best among all the online algorithms and is quite close to the offline optimal results when $|T|(|W) = 10K$ and 20K. As for running time, we can see that Greedy is the most efficient and HST-Greedy is nearly as good as Greedy as both algorithms take $O(|W|)$ time to process each new-coming user. However, HST-Reassignment is highly inefficient due to is $O(|W|^2)$ time complexity. As for memory consumption, Greedy is again the most efficient since no extra storage for the HST structure is needed. In overall, we can see that Greedy is much more efficient and scalable in both time and space than the other state-of-art online algorithms.

Comparisons with Permutation. The results of the comparison with Permutation on a smaller dataset are presented in Table 4. We also show the results of the other algorithms. For total distance, we can observe that Permutation is always worse than Greedy but is better than the other two online algorithms. However, according to the competitive analysis under the adversarial model, the ranking of the algorithms in descending order of their competitive ratios is that Greedy $\gg$ Permutation $>$ HST-Greedy $>$ HST-Reassignment. It indicates that the competitive ratio under the adversarial model can no way reflect the real performance of an algorithm in practice. As for running time, Permutation is highly inefficient as it took hundreds of seconds to return an assignment while all the other algorithms return results in less than a second. Permutation is also less efficient in space than the other algorithms.

Real dataset. The results on real dataset are presented in the last column of Figure 11. For total distance, we can again similar results that Greedy performs better than the other two online algorithms and is only slightly worse than the offline optimal algorithm. Particularly, Greedy is almost 2 times better than the other two online algorithms. Also, an interesting observation is that the total distances generated by all the algorithms are quite large around 12-18PM, and are lowest around 0-6AM, conforming to the statistics of the dataset that there are more taxi-calling tasks from 12-18PM

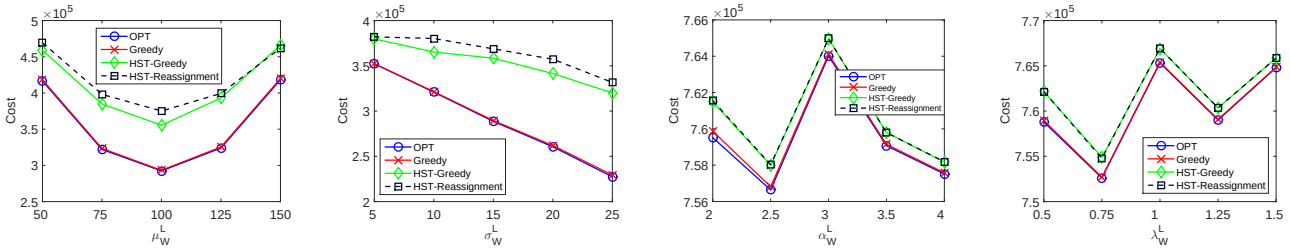


(a) Cost of varied Normal μ_W^L (b) Cost of varied Normal σ_W^L (c) Cost of varied Power α_W^L (d) Cost of varied Exp λ_W^L

(e) Time of varied Normal μ_W^L (f) Time of varied Normal σ_W^L (g) Time of varied Power α_W^L (h) Time of varied Exp λ_W^L

(i) Memory of varied Normal μ_W^L (j) Memory of varied Normal σ_W^L (k) Memory of varied Power α_W^L (l) Memory of varied Exp λ_W^L

Figure 8: Results that the locations of service providers in W follow Normal, Power-law, and Exponential distributions while the locations of users T follow Exponential distribution.



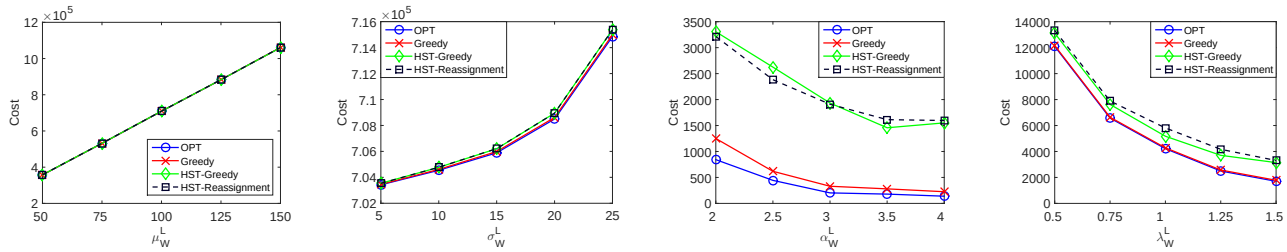
(a) Cost of varied Normal μ_W^L (b) Cost of varied Normal σ_W^L (c) Cost of varied Power α_W^L (d) Cost of varied Exp λ_W^L

Figure 9: Results that the locations of service providers in W follow Normal, Power-law, and Exponential distributions while the locations of users in T follow Uniform distribution.

than from 0-6AM. As for running time and memory results, we can see that Greedy is still the most efficient in both time and space, and HST-Reassignment is the most inefficient.

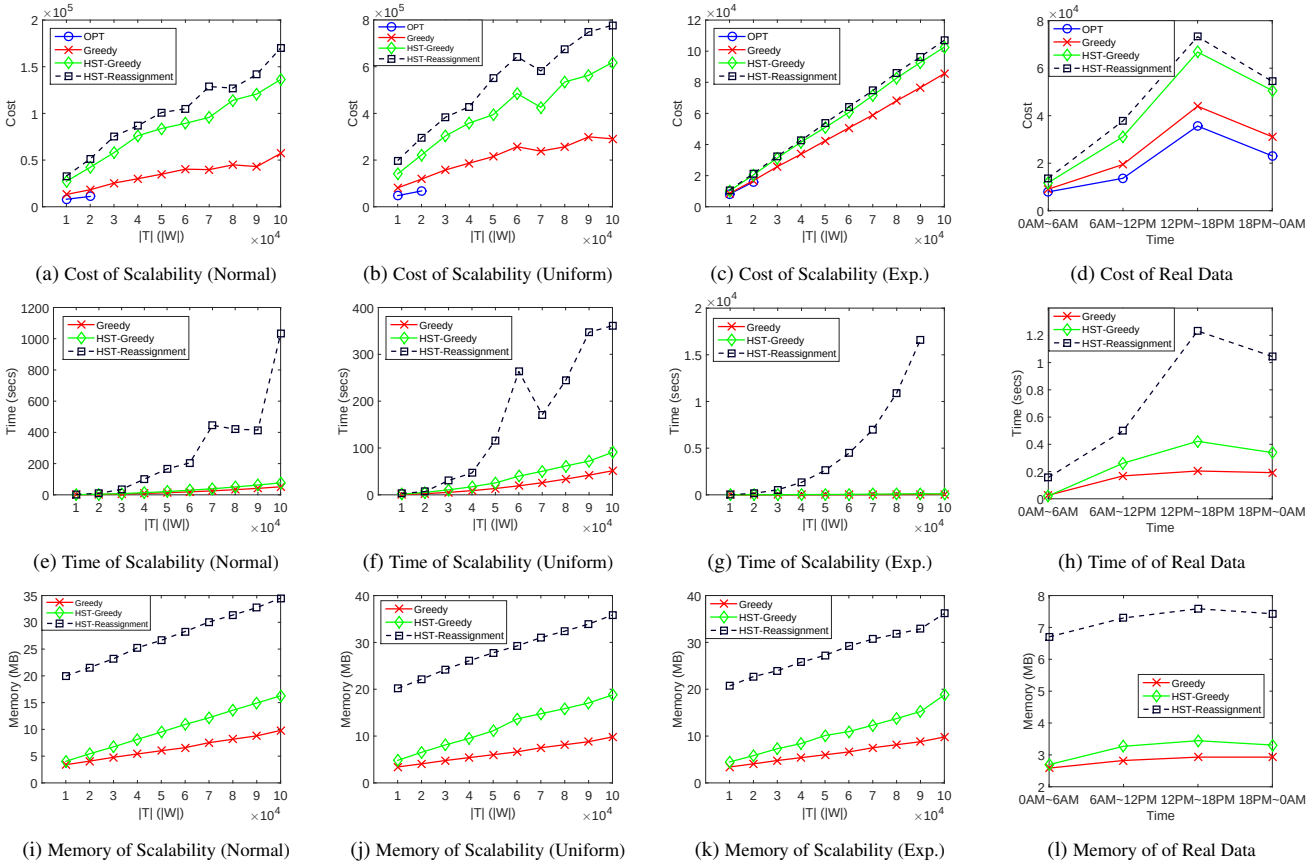
5.3 Summary

- Greedy generates total distance that is at most two times of offline optimal algorithm in all the experiments on real data and all different distributions of synthetic data. Therefore, we propose the hypothesis that Greedy has constant competitive ratio under the random order model when locations of users and service providers follow any combination of the Uniform, Normal, Power-law and Exponential distributions. We further hypothesize that Greedy has constant competitive ratio under the random order model in general.
- According to the competitive analysis of the algorithms under the adversarial model, the ranking of the algorithms in descending order of their competitive ratios is that Greedy $\gg$ Permutation $>$ HST-Greedy $>$ HST-Reassignment. However, the extensive experiments show that Greedy is the best in practice while HST-Reassignment is the worst. It indicates that the competitive ratio of an algorithm under the adversarial model cannot reflect the real performance of the algorithm in practice, and we should not only focus on improving the worst-case performance of an online algorithm.
- Greedy performs the best. Particularly, Greedy is sometimes at least two times better than the other online algorithms when the size of data is smaller, and is even several times



(a) Cost of varied Normal μ_W^L (b) Cost of varied Normal σ_W^L (c) Cost of varied Power α_W^L (d) Cost of varied Exp λ_W^L

Figure 10: Results that the locations of service providers in W follow Normal, Power-law, and Exponential distributions while the locations of users in T follow Power-law distribution.



(a) Cost of Scalability (Normal) (b) Cost of Scalability (Uniform) (c) Cost of Scalability (Exp.) (d) Cost of Real Data

(e) Time of Scalability (Normal) (f) Time of Scalability (Uniform) (g) Time of Scalability (Exp.) (h) Time of of Real Data

(i) Memory of Scalability (Normal) (j) Memory of Scalability (Uniform) (k) Memory of Scalability (Exp.) (l) Memory of of Real Data

Figure 11: Results on scalability test and real dataset

better when the size of data scales as the scalability test indicates.

- Greedy is more efficient in time than other the online algorithms and consumes least space among all the online algorithms, and HST-Greedy is slightly inefficient than Greedy in terms of running time.

6. CONCLUSION AND OPEN QUESTION

In this paper, we conduct a comprehensive experimental study for the online minimum bipartite matching in real time spatial data (OMBM) problem through evaluating four representative online algorithms, i.e. Greedy, Permutation, HST-Greedy, and HST- Reassignment, on five real and synthetic datasets with different characteristics. We provide efficient and uniform implementations of four existing representative algorithms, and obtain the following three experimental findings and propose an open question.

First, our most important experimental finding is that both the efficiency and the effectiveness of Greedy significantly outperforms the other algorithms in almost all practical cases though Greedy has been always considered as the worst algorithm in past 25 years due to its exponential competitive ratio under the adversarial model (the worst-case analysis). In particular, the worst case in the adversarial model of Greedy has constant competitive ratio, 3.195, in the random order model (the average-case analysis). In summary, we try to clarify the 25-year misunderstanding towards Greedy for the OMBM problem through the experimental study.

Second, existing studies for the OMBM problem believe that online algorithms with smaller competitives ratio have the better per-

formance. Then according to the ascending order of the competitive ratios of the algorithms compared under the adversarial model, we have HST-Reassignment $<$ HST-Greedy $<$ Permutation $\ll$ Greedy. However, the extensive experiments show that the ranking of these algorithms in terms of effectiveness is quite different in practice - Greedy performs the best. It indicates that the competitive analysis under the adversarial model cannot reflect the real performance of an online algorithm in practice. Therefore, it suggests that we should not only focus on improving the worst-case performance of an online algorithm but should pay more attention to its average-case performance.

Third, HST-Greedy is the runner-up. Particularly, since HST-Greedy relies on the HST structure, which introduces extra projection errors, HST-Greedy performs worse than Greedy in overall. However, as HST-Greedy adopts the greedy strategy, it is still much more effective than HST-Reassignment though HST-Reassignment has better competitive ratio under the adversarial model in theory.

Finally, though we still cannot prove that the competitive ratio of Greedy in the average-case analysis is a constant, the aforementioned extensive random experiment results motivate us to propose the following hypothesis as a open question: *the average-case competitive ratio under the random order model of Greedy for the OMBM problem should be constant*, which can provide a theoretical explanation for the outstanding performance of Greedy in practice if the hypothesis holds.

Acknowledgment

We are grateful to anonymous reviewers for their constructive comments on this work. This work is supported in part by the National Science Foundation of China (NSFC) under Grant No. 61502021, 61328202, 71531001, National Grand Fundamental Research 973 Program of China under Grant 2014CB340300, the Hong Kong RGC Project N_HKUST637/13, NSFC Guang Dong Grant No. U1301253, Microsoft Research Asia Collaboration Research Grant, Google Faculty Award 2013, and Microsoft Research Asia Fellowship 2012.

7. REFERENCES

[1] Gigwalk. *http://www.gigwalk.com*.

[2] Grubhub. *https://www.grubhub.com/*.

[3] Shenzhou private cars. *http://zhuanche.zuche.com/*.

[4] Source code and datasets. *https://www.cse.ust.hk/ jshe/OMBM.zip*.

[5] Uber. *https://www.uber.com/*.

[6] R. Ahuya, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[7] A. Alfarrarjeh, T. Emrich, and C. Shahabi. Scalable spatial crowdsourcing: A study of distributed algorithms. In *MDM 2015*.

[8] N. Bansal, N. Buchbinder, A. Gupta, and J. S. Naor. An o ($\log^2$k)-competitive algorithm for metric bipartite matching. In *ESA 2007*.

[9] N. Bansal, N. Buchbinder, A. Gupta, and J. S. Naor. A randomized o ($\log^2$k)-competitive algorithm for metric bipartite matching. *Algorithmica*, 2014.

[10] R. E. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems, Revised Reprint*. 2009.

[11] Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, Y. Tong, and C. J. Zhang. gmission: A general spatial crowdsourcing platform. *PVLDB 2014*.

[12] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC 2003*.

[13] J. Gao, L. Guibas, N. Milosavljevic, and D. Zhou. Distributed resource management and matching in sensor networks. In *IPSN 2009*.

[14] A. Gupta and K. Lewi. The online metric matching problem for doubling metrics. In *ICALP 2012*.

[15] Z. Jiang, W. Xie, M. Li, B. Podobnik, W. Zhou, and H. E. Stanley. Calling patterns in human communication dynamics. *Proceedings of the National Academy of Sciences*, 2013.

[16] B. Kalyanasundaram and K. Pruhs. On-line weighted matching. In *SODA 1991*.

[17] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 1993.

[18] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *GIS 2012*.

[19] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 1994.

[20] D.-H. Lee, H. Wang, R. Cheu, and S. Teo. Taxi dispatch system based on current demands and real-time traffic conditions. *Transportation Research Record: Journal of the Transportation Research Board*, 2004.

[21] X. Liang, J. Zhao, L. Dong, and K. Xu. Unraveling the origin of exponential law in intra-urban human mobility. *Scientific reports*, 2013.

[22] A. Meyerson, A. Nanavati, and L. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *SODA 2006*.

[23] K. T. Seow, N. H. Dang, and D.-H. Lee. A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation Science and Engineering*, 2010.

[24] J. She, Y. Tong, and L. Chen. Utility-aware social event-participant planning. In *SIGMOD 2015*.

[25] J. She, Y. Tong, L. Chen, and C. C. Cao. Conflict-aware event-participant arrangement. In *ICDE 2015*.

[26] J. She, Y. Tong, L. Chen, and C. C. Cao. Conflict-aware event-participant arrangement and its variant for online setting. *IEEE Transactions on Knowledge and Data Engineering, 2016*.

[27] H. To, G. Ghinita, and C. Shahabi. A framework for protecting worker location privacy in spatial crowdsourcing. *PVLDB 2014*.

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

[29] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen. Online mobile micro-task allocation in spatial crowdsourcing. In *ICDE 2016*.

[30] Y. Tong, J. She, and R. Meng. Bottleneck-aware arrangement over event-based social networks: the max-min approach. *World Wide Web Journal, 2016*.

[31] L. H. U, M. L. Yiu, K. Mouratidis, and N. Mamoulis. Capacity constrained assignment in spatial databases. In *SIGMOD 2008*.

[32] R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao. On efficient spatial matching. In *VLDB 2007*.

[33] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence, 2007*.