IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 11, 2020, accepted March 13, 2020, date of publication March 23, 2020, date of current version April 1, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2982423

Spatial Two-Sided Online Bottleneck Matching With Deadlines

LONG LI AND WEIFENG LV
SKLSDE Lab and BDBC, Beihang University, Beijing 100091, China
Corresponding author: Weifeng Lv (lwf@buaa.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant U11811463, and in part by the Science and Technology Major Project of Beijing under Grant Z191100002519012.

ABSTRACT Recently, there are several studies focusing on the bottleneck optimization objective in Spatial Crowdsourcing (SC). However, these studies usually do not consider the deadline constraint. Different from these studies, we take deadlines into consideration and identify the Fully Online Bottleneck Matching with Deadlines (FOBMD) problem in SC. Because of the deadlines, consideration must be given to both the bottleneck cost and the cardinality, which makes the FOBMD problem more challenging, and no online algorithm without actively refusing tasks can achieve a constant competitive ratio of the bottleneck cost for the FOBMD problem. To settle the FOBMD problem, we consider three baseline algorithms and propose an online algorithm, namely Local Isolated Point Greedy (LIPG). Finally, we validate the effectiveness and efficiency of our proposed algorithm via extensive experiments on both synthetic and real world datasets.

INDEX TERMS Spatial crowdsourcing, online bottleneck matching, competitive ratio analysis.

I. INTRODUCTION

Applications and Services based on Spatial Crowdsourcing (SC) become very popular with the development of smart phones. These services of SC, such as Uber,[1] Lyft[2] and DiDi,[3] really make people's life more convenient. However, one of the main challenges in SC is how to assign tasks released by customers to workers [1], [2], especially when tasks or workers dynamically appear on the SC platform in real-world scenarios.

Existing studies [3]–[11] model the task-worker assignment problem in SC as an online matching problem for a specifical optimization objective, *e.g.* maximizing the number of performed tasks (cardinality) [3], [4], maximizing the total utility of the assignment [5]–[8], minimizing the total cost to perform all tasks [9], [12] and the bottleneck optimization objective, namely minimizing the maximum cost of all taks-worker matching pairs [10], [11]. The bottleneck optimization objective is an emerging optimization objective for online matching problems in SC and the existing studies [10], [11] all focus on the two-sided online scenario, where the both tasks and workers dynamically appear on the platform. More importantly, [10], [11] do not consider the deadline constraint and they both assume tasks or workers can wait until the algorithms matching them. However, this assumption is unrealistic and unreasonable in real-world scenarios. Taking the online taxi-calling service as an example, customers dynamically log in the SC platform and submit their orders (tasks). these tasks can be constrained by deadlines, namely customers would be impatient and cancel their orders if the platform does not assign orders to workers (taxis) in a short period of waiting time. For the benefit of SC platforms and better users' experience, we should take the deadline constraint into consideration and study the online bottleneck matching problem with the deadline constraint in SC.

More generally, we consider a situation where both tasks and workers are constrained by deadlines, namely both tasks and workers should be matched before their deadlines, or otherwise expire after deadlines. Due to the existence of deadlines, a contradiction between the bottleneck cost and cardinality arises. Straightly speaking, the more tasks we actively refuse, the bottleneck cost, namely the maximum cost of all task-worker matching pairs, is more likely to be lower. And extremely, the bottleneck cost can be zero if we refuse all released tasks and the cardinality of the assignment is also zero. However, actively refusing the customers' orders (tasks) is ridiculous and unacceptable in real-world scenarios. And we must consider the cardinality optimization

The associate editor coordinating the review of this manuscript and approving it for publication was Yongqiang Zhao.

[1]https://www.uber.com/
[2]https://www.lyft.com/
[3]https://www.didiglobal.com/

objective at first and then the bottleneck optimization objective, namely maximizing the number of task-worker matching pairs and minimizing the maximum cost of all task-worker matching pairs.

In this paper, we study the Fully Online Bottleneck Matching with Deadlines (FOBMD) problem in SC. Specifically, given a set of spatial workers and a set of spatial tasks, both tasks and workers dynamically appear on the platform and their spatial locations are unknown until their appearance, and both tasks and workers should be matched before their deadlines, the FOBMD problem is to find a matching between tasks and workers with maximizing the cardinality and minimizing the bottleneck cost.

The FOBMD problem is hard to settle and even more harder than the two-sided online bottleneck matching problems studied in [10], [11] due to consideration of both the cardinality and bottleneck cost. Firstly, the bottleneck cost is very sensitive and even a bad task-worker matching pair can make the final bottleneck cost unusual high. Secondly, we have to apply some special strategies to resist sensibility of bottleneck cost and lower the bottleneck cost, however such strategies would have some bias for some types of tasks/worker and can lead to lower cardinality. The tradeoff between minimizing the bottleneck cost and maximizing the cardinality in the FOBMD problem is a new challenge when comparing with the two-sided online bottleneck matching problems studied in [10], [11].

To handle the FOBMD problem, we propose an online algorithm, namely Local Isolated Point Greedy (LIPG). The core idea of LIPG is that we assume the tasks/workers surrounded by less workers/tasks are more likely to result in the larger bottleneck cost and we greedily match a task/worker to its isolated neighbor within a local range.

We summarize our main contributions as follows:

- We identify a two-sided online bottleneck matching problem with the deadline constraint in SC, the Fully Online Bottleneck Matching with Deadlines (FOBMD) problem, as a first work considering both minimizing bottleneck cost and maximizing the cardinality of two-sided online bottleneck matching in SC. And we clarify that no online algorithms without actively refusing tasks can achieve a constant competitive ratio of the bottleneck cost for the FOBMD problem.
- We clarify that the existing online algorithms for solving the two-sided online maximum cardinality matching problem in SC or two-sided online bottleneck matching problem without deadlines in SC, Greedy, Ranking and Batch, do not work nicely for the FOBMD problem.
- We propose an online algorithm, namely Local Isolated Point Greedy (LIPG), to solve the FOBMD problem.
- The effectiveness and efficiency of our proposed algorithm is verified through extensive experiments on both synthetic and real datasets.

The rest of the paper is organized as follows. In Section III, we formally formulate the FOBMD problem and the competitive ratio, and then we give some competitive ratio analysis for the FOBMD problem. In Section II, we review related work. In Section IV, we review three existing state-of-the-art existing the two-sided online maximum cardinality matching problem in SC or two-sided online bottleneck matching problem without deadlines in SC, and then we introduce our proposed algorithm. Extensive experiments on both synthetic and real-world datasets are presented in Section V. In Section VI, we conclude the paper.

II. RELATED WORK

In this section, we review related work from two categories: online matching in spatial crowdsourcing, general fully online matching with deadlines, and bottleneck bipartite matching.

A. ONLINE MATCHING IN SPATIAL CROWDSOURCING

There are a lot of existing studies focusing on the online matching problem in SC [2]–[13]. Most of the existing studies consider the maximum-cardinality optimization objective [3], [4], [13], [14], or the max-/min-sum-cost optimization objective [5]–[9], [12], [15]–[19]. And only two related studies [10], [11] consider the minimum bottleneck cost optimization objective as we consider in this paper. However, [10], [11] do not consider the deadline constraint and they just make the tasks/workers waiting as long as they want. In fact, there exists the deadline constraint in real-world scenarios. And we consider the deadline constraint which is different which other existing studies.

B. GENERAL FULLY ONLINE MATCHING WITH DEADLINES

Recently, there are some studies focusing on the general fully online matching problem with deadlines in the theoretical computing field [20]–[23]. References [20], [21] focus the optimization objective of maximizing the cardinality and [22], [23] focus the optimization objective of maximizing the total utility of all task-worker matching pairs. All these studies do not consider the bottleneck optimization objective and this is the difference between our paper and these existing studies.

C. BIPARTITE BOTTLENECK MATCHING

The offline bipartite bottleneck matching problem is been introduced in 1953 [24] and then studied by a lot of related work [25]–[30]. And these studies do not work for the FOBMD problem due to the online setting where tasks and workers appear online and the information of tasks and worker is unknown before appearing.

However, there are only a few studies [31]–[34] considering the bipartite bottleneck matching problem in the one-sided online scenario where all workers are known in advance and the tasks appear online one-by-one, and this setting is different with our paper where both tasks and workers appear online.

III. PRELIMINARIES

In this section, we formally define the FOBMD problem and the standard competitive ratio (CR). And then, we give the basic competitive ratio analysis for the FOBMD problem.

A. PROBLEM DEFINITION

Definition 1 (Task): A spatial task ("task" for short), denoted by $t =< l_t, a_t, d_t >$*, is located at* l_t *in a 2D space, appears on the platform at time* a_t *with the deadline* d_t *and needs to be answered before time* $a_t + d_t$*, otherwise it will expire.*

Definition 2 (Worker): A spatial worker ("worker" for short), denoted by $w =< l_w, a_w, d_w >$*, is located at* l_w *in a 2D space and appears on the platform at time* a_w *with the deadline* d_w *and needs to be answered before time* $a_w + d_w$*, otherwise it will expire.*

Definition 3 (The FOBMD Problem): Given a set of tasks T *and a set of workers* W*, where both tasks and workers dynamically appear on the SC platform and their spatial information is unknown before they appear, and given a metric distance function* $dis(.,.)$ *in 2D space, the Fully Online Bottleneck Matching with Deadlines (FOBMD) problem is to find a matching* M *to maximize the number of task-worker matching pair* $|M|$ *and minimize the maximum distance cost of all task-worker matching pairs,* $Cost(M) = \max_{t\in T, w\in W} dis(t, w)$*. such that the following constraints are satisfied*:

- *Deadline Constraint: each task/worker should be matched after appearing and before its deadline, or otherwise it will expire and remain unmatched after the deadline.*
- *Capacity Constraint: a task can be assigned to only one worker and vice versa.*
- *Invariable Constraint: once a task is assigned to a worker, the corresponding task-worker matching pair cannot be revoked.*

B. COMPETITIVE ANALYSIS MODELS

The Competitive Ratio (CR), namely the worst-case ratio of an online algorithm's result to the optimum over all possible inputs and all possible arrival orders, is a standard evaluation for online algorithms. The definition of the competitive ratio of the bottleneck cost and the competitive ratio of the cardinality is shown as follows.

Definition 4 (Competitive Ratio of the Bottleneck Cost): The competitive ratio in the adversarial model of the bottleneck cost for the FOBMD problem is defined as follows:

$$CR(Bottleneck)_A = \max_{\forall T, \forall W, \sigma_T, \forall \sigma_W \forall} \frac{Cost(M)}{Cost(OPT)} \quad (1)$$

where T *and* W *is an arbitrary set of tasks and workers respectively,* σ_T *is an arbitrary arrival order of the tasks* T*,* σ_W *is an arbitrary arrival order of the workers* W*,* $Cost(M)$ *is the bottleneck cost generated by the online algorithm, and* $Cost(OPT)$ *is the offline optimal bottleneck cost.*

Note that the aforementioned $Cost(OPT)$ can be calculated by classical offline bipartite bottleneck matching algorithms, *e.g.* the threshold-based algorithms [25], [26], [28], [29] and the swap-chain algorithm [30].

Definition 5 (Competitive Ratio of the Cardinality): The competitive ratio in the adversarial model of the cardinality for the FOBMD problem is defined as follows:

$$CR(Cardinality)_A = \max_{\forall T, \forall W, \sigma_T, \forall \sigma_W \forall} \frac{|M|}{|OPT|} \quad (2)$$

where T *and* W *is an arbitrary set of tasks and workers respectively,* σ_T *is an arbitrary arrival order of the tasks* T*,* σ_W *is an arbitrary arrival order of the workers* W*,* $|M|$ *is the cardinality of the matching output by the online algorithm, and* $|OPT|$ *is the cardinality of the offline optimal bottleneck cost, where* $|OPT|$ *is also the cardinality of the offline maximum matching of* T *and* W*.*

C. COMPETITIVE ANALYSIS

We give some basic competitive analysis in this subsection.

Example 1: Consider all tasks and workers located on a line, and the locations of $k + 1$ *tasks,* $\{t_0, t_1, \ldots, t_k\}$*, are points* $\{2^0, 2^1, \ldots, 2^k\}$ *respectively. And* k *workers,* $\{w_0, w_1 \ldots, w_{k-1}\}$*, locate at points* $\{2^1+\epsilon, 2^2+\epsilon \ldots, 2^k+\epsilon\}$ *respectively, where* ϵ *is an arbitrarily small positive number. And the arrival time of* $k + 1$ *tasks and* k *workers are* $\{1, 3, 5, \ldots, 2k + 1\}$ *and* $\{2, 4, 6, \ldots, 2k\}$*, respectively. The deadlines of all tasks and workers are 2. The arrival time of tasks and workers on the time line is illustrated in Fig. 1.*

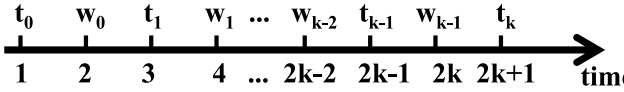


FIGURE 1. An illustration of the arriving time of task and workers on the time line.

Theorem 1: No online algorithm without actively refusing tasks can achieve a constant competitive ratio of the bottleneck cost for the FOBMD problem.

Proof: As shown in Example 1, apparently, the only worker available for task t_0 before t_0's deadline is w_0. If the online algorithm does not actively refuse any tasks, the online algorithm has to match t_0 to w_0 and finally outputs a result with $\{t_0, t_1, \ldots, t_{k-1}\}$ being matched to $\{w_0, w_1, \ldots, w_{k-1}\}$, respectively, and the cardinality is k, the bottleneck cost is $Cost(M) = 2^{k-1} + \epsilon$. The optimal algorithm would match $\{t_1, t_2, \ldots, t_k\}$ to $\{w_0, w_1, \ldots, w_{k-1}\}$, respectively, and the cardinality is k and the optimal bottleneck cost is $Cost(OPT) = \epsilon$. And we have:

$$CR(Bottleneck) = \frac{Cost(M)}{Cost(OPT)} = \frac{2^{k-1} + \epsilon}{\epsilon} \to \infty \quad (3)$$

In this case, the competitive ratio of any online algorithm without actively refusing tasks approaches infinity. Obviously, there is no online algorithm without actively refusing tasks can achieve a constant competitive ratio of the bottleneck cost for the FOBMD problem. □

Theorem 2 [21]: The lower bound and upper bound of the competitive of the cardinality is 0.5 and 0.6317, respectively.

Reference [21] has discussed the bound of competitive ratio for the two-sided online maximum matching problem with deadlines. And the two-sided online maximum matching problem only considers the cardinality optimization objective and the conclusion also applies to the FOBMD problem for the competitive ratio of the cardinality.

The aforementioned theorem shows that the FOBMD is hard if considerations are given to both the cardinality and the bottleneck cost. For a better competitive ratio of cardinality and trying to service every customer, the competitive ratio of the bottleneck cost can be as bad as infinity. There is no upper bound for the competitive ratio of the bottleneck cost.

IV. ALGORITHMS

In this section, we present the baseline algorithms and our proposed algorithm.

A. BASELINE ALGORITHMS

Existing online algorithms only consider one optimization objective, the bottleneck cost (the two-sided online bottleneck matching problem without deadlines) or the cardinality (the two-sided online maximum matching problem). The existing online algorithms for the two-sided online bottleneck cost matching problem without deadlines [10], [11] do not apply for the FOBMD problem due to the deadline constraints. Especially, the proposed algorithm based on HST (Hierarchically well-Separated Tree) [11] needs to know all the locations of workers beforehand for constructing the HST and is not appropriate for the FOBMD problem.

For clarifying that these existing online algorithms solving the two-sided online maximum matching problem do not work for the FOBMD problem. We consider three representative or state-of-the-art algorithms for the two-sided online maximum matching problem as baseline algorithms, Greedy, Ranking and Batch.

1) GREEDY ALGORITHM

Greedy [7], [35], [36]is the most straightforward online algorithm for solving online matching problems and the main idea of Greedy is to match tasks/workers to their nearest available neighbor worker/task as soon as the tasks/workers arrive.

Example 2: Consider k tasks and k workers locating on a line, and the k tasks, $\{t_0, t_1, \ldots, t_{k-1}\}$, arrive at time $\{1, 3, \ldots, 2k-1\}$ and locate points $\{1, 2^1, \ldots, 2^{k-1}\}$, respectively. And the k workers, $\{w_0, w_1, \ldots, w_{k-1}\}$, arrive at time $\{2, 4, \ldots, 2k\}$ and locate at points $\{-\epsilon, 2^1, \ldots, 2^{k-1}\}$ respectively, where ϵ is an arbitrarily small positive number. The deadlines of all tasks and workers are the same, 10.

As shown in the aforementioned example, Greedy outputs a result with $\{t_0, t_1, \ldots, t_{k-1}\}$ respectively matching to $\{w_0, w_1, \ldots, w_{k-1}\}$, the cardinality is k and the bottleneck cost is $1 + \epsilon$.

Complexity Analysis: The time and space complexity of Greedy are $O(max(|T'|, |W'|))$ for each task/worker achieving its deadline, where $|T'|$ and $|W'|$ are the number of available tasks and workers when matching.

2) RANKING ALGORITHM

References [20], [37] show the randomized ranking algorithm, namely "Ranking", performs nicely for the two-sided online maximum matching with deadlines and the competitive ratio for the cardinality of the Ranking algorithm is between 0.5541 and 0.5671 [20]. To verify the performance of Ranking for the FOBMD problem, we use Ranking as a baseline algorithm in this paper.

The **basic idea** of Ranking is simple: each task or worker uniformly picks a random float value in interval [0, 1), and a task/worker is assigned to the unmatched worker/task with the minimum random float value when the task/worker reaches its deadline. The produce of Ranking is shown in Alg. 1.

Algorithm 1 Ranking Algorithm [20]

input : The task set T, the worker set W
output: The matching M between T and W
(1) a task or worker v arrives:
pick $y_v \in [0, 1)$ uniformly at random;
(2) a task or worker v reaches its deadline:
let $N(v)$ be the set of unmatched neighbors of v;
if $N(v) = \emptyset$ **then**
    v remains unmatched;
else
    $u' \leftarrow \arg\min_{u \in N(v)} y_u$;
    $M \leftarrow M \cup \{(u', v)\}$;
return M;

Back to Example 2, Ranking is a randomized algorithm and Ranking randomly matches t_0 to one of $\{w_0, w_1, w_2, w_3, w_4\}$ when t_0 reaches its deadline. And Ranking outputs a result with the cardinality k and the bottleneck cost which is at least $1 + \epsilon$ and at most $2^{k-1} - 2^{k-5} = 15 \cdot 2^{k-5}$.

Complexity Analysis. The time and space complexity of Ranking are the same with the aforementioned Greedy's.

3) BATCH ALGORITHM

Batch is proposed in [38] and is demonstrated as the state-of-art algorithm for the two-sided online maximum matching with deadlines in SC [14]. We slightly adjust Batch a bit to fit the FOBMD problem by calculating the optimal bottleneck matching in each batch. And we still refer the slightly adjusted Batch as "Batch" and use it as a baseline algorithm.

The **basic idea** of Batch is to divide the timeline into a number of discrete batches and treat the matching problem in each batch as an instance of offline bottleneck matching problem. Specifically, the arriving tasks/workers must wait until they reach the end of a batch and the tasks/workers in each batch are matched according the optimal bottleneck matching result of these tasks and workers. The produce of

Algorithm 2 Batch Algorithm

input : The task set T, the worker set W
output: The matching M between T and W

(1) A task t arrives:
$T \leftarrow t \cup T$;
(2) A worker w arrives:
$W \leftarrow w \cup W$;
(3) Matching procedure:
update T and W by deleting the expired tasks/workers;
let the current timestamp be ζ;
let T'/W' be the tasks/workers arriving in time $[\zeta - \theta, \zeta]$;
calculate the optimal bottleneck matching M' of T' and W';
$M \leftarrow M \cup M'$;
update T and W by deleting the already matched tasks/workers;
return M;

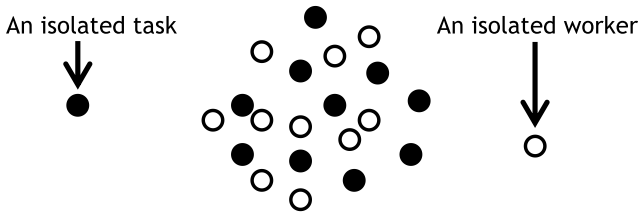


FIGURE 2. An illustration for the isolated task and the isolated worker.

BATCH is illustrated in Alg. 2 and θ is the length setting of batch.

Back to Example 2, we assume that $\theta = 10$ and Batch matches tasks and workers every $\theta = 10$ timestamps. At time 10, Batch matches $\{t_0, t_1, t_2, t_3\}$ to $\{w_0, w_1, w_2, w_3\}$, respectively. Finally, Batch outputs a result with matching $t_0, t_1, t_3, \ldots, t_{k-1}$ to $w_0, w_1, w_2, \ldots, w_{k-1}$, respectively. The bottleneck cost is $1 + \epsilon$ and the matching cardinality is k.

Complexity Analysis: Comparing with the aforementioned algorithms, Greedy and Ranking, Batch is more complex and time-consuming because Batch has to calculate of the optimal bottleneck matching in each batch. For each batch, the space complexity of Batch is $O(|T'| \cdot |W'|)$ and the time complexity is $O(log(max(|T'|, |W'|)) \cdot \alpha)$ where α is the cost of a maximum flow algorithm (*e.g.* $\alpha = O((|T'| \cdot |W'|)^2 \cdot N)$) on a flow network with $|T'| \cdot |W'|$ vertices and N edges if we use the proposed IBFS algorithm in [39], where T' and W' are the task set and worker set within the batch, respectively.

B. OUR PROPOSED ALGORITHM

In this subsection, we introduce our proposed online algorithm for solving FOBMD, called Local Isolated Point Greedy (LIPG).

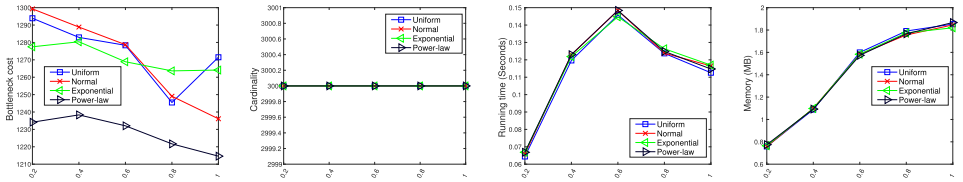
1) CORE IDEA

There are many reasons can lead to the large bottleneck cost and we try to overcome one of the reasons, isolated tasks/workers, by designing an online algorithm and solving the FOBMD problem. As shown in Fig. 2, there are some tasks and workers located on a 2D space, where the solid

Algorithm 3 LIPG Algorithm

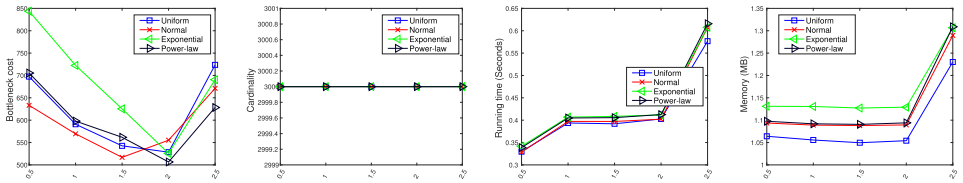
input : The task set T, the worker set W
output: The matching M between T and W

(1) A task t arrives:
$T \leftarrow t \cup T$;
(2) A worker w arrives:
$W \leftarrow w \cup W$;
(3) Matching procedure:
update T and W by deleting expired tasks/workers;
$\Delta \leftarrow \sum_{i=1}^{i=|T|} \sum_{j=1}^{j=|W|} dis(t_i, w_j)$;
$\delta \leftarrow \frac{\Delta}{|T| \cdot |W|}$;
$TLD(T) \leftarrow [0, 0, \ldots, 0]_{|T|}$;
$TLD(W) \leftarrow [0, 0, \ldots, 0]_{|W|}$;
foreach t_i *in* $|T|$ **do**
  $W' \leftarrow \{\forall w | w \in W$ *and* $dis(t_i, w) \leq \eta \times \delta\}$;
  $TLD(t_i) \leftarrow |W'|$;
foreach w_j *to* $|W|$ **do**
  $T' \leftarrow \{\forall t | t \in T$ *and* $dis(t, w_j) \leq \eta \times \delta\}$;
  $TLD(w_j) \leftarrow |T'|$;
foreach t_i *in* $|T|$ **do**
  if t_i *reaches its deadline* **then**
    if $TLD(t_i) = 0$ **then**
      $w' \leftarrow \arg\min_{w \in W} dis(t_i, w)$;
    else
      $\mathbb{CW} \leftarrow \{\forall w | w \in W$ *and* $dis(t_i, w) \leq \kappa \times \delta\}$;
      $w' \leftarrow$the worker in $\mathbb{CW}$ with largest *TLD*; //If there are multiple workers with the same largest *TLD*, the algorithm chooses the one with the smallest distance to t_i as w';
    $M \leftarrow M \cup (t_i, w')$;
    $W \leftarrow W - w'$;
update T by deleting the already matched tasks;
foreach w_j *to* $|W|$ **do**
  if w_j *reaches its deadline* **then**
    if $TLD(w_j) = 0$ **then**
      $t' \leftarrow \arg\min_{t \in T} dis(t, w_j)$;
    else
      $\mathbb{CT} \leftarrow \{\forall t | t \in T$ *and* $dis(t, w_j) \leq \kappa \times \delta\}$;
      $t' \leftarrow$the task in $\mathbb{CT}$ with largest *TLD*; //If there are multiple tasks with the same largest *TLD*, the algorithm chooses the one with the smallest distance to w_j as t';
    $M \leftarrow M \cup (t', w_j)$;
    $T \leftarrow T - t'$;
update W by deleting the already matched workers;
return M;



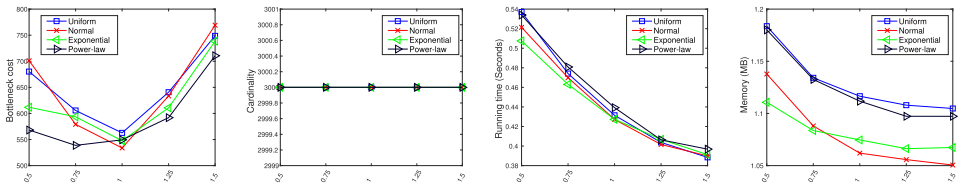
(a) Bottleneck cost (b) Matching cardinality (c) Running time (d) Memory

FIGURE 3. Results of varied θ of Batch when tasks and workers follow different distributions.



(a) Bottleneck cost (b) Matching cardinality (c) Running time (d) Memory

FIGURE 4. Results of varied η of LIPG when tasks and workers follow different distributions.



(a) Bottleneck cost (b) Matching cardinality (c) Running time (d) Memory

FIGURE 5. Results of varied κ of LIPG when tasks and workers follow different distributions.

points indicate tasks and the hollow points indicate workers. The task/worker locating far away other workers/tasks and surrounded by less workers/tasks is isolated. Obviously, the isolated task/worker is more likely to result in the large bottleneck cost if the matching strategy is inappropriate. And we can lower the bottleneck by rightly handling the isolated tasks/workers. In the two-sided online scenario, we do not know spatial information of future tasks/worker, and the right choice is to preferentially match the current isolated tasks/workers in case the isolated tasks/workers become more isolated in the future and end up with a worse matching.

The remaining question is how to identify the isolated tasks/workers. Inspired by a density-based outlier detection algorithm, LoF [40], we define a metric, namely Thresholded Local Density (TLD), to identify whether a task/worker is isolated or not. It turns out that the TLD works nicely for the FOBMD problem and the definition of TLD is shown as follows.

Definition 6 (Thresholded Local Density): Given a set of tasks $T = \{t_0, t_1, \ldots, t_{k-1}\}$ *and workers* $W = \{w_0, w_1, \ldots, w_{k-1}\}$ *with specific locations in a 2D space, a metric distance function* $dis(.,.)$ *and a distance threshold* φ ($\varphi > 0$)*, the Thresholded Local Density (TLD) of an task* t_i *is*

$$TLD(t_i) = |W'| \tag{4}$$

where $W' = \{\forall w \in W | dis(t_i, w) \leq \varphi\}$.

Similarly, the Thresholded Local Density (TLD) of an worker w_j *is*

$$TLD(w_j) = |T'| \tag{5}$$

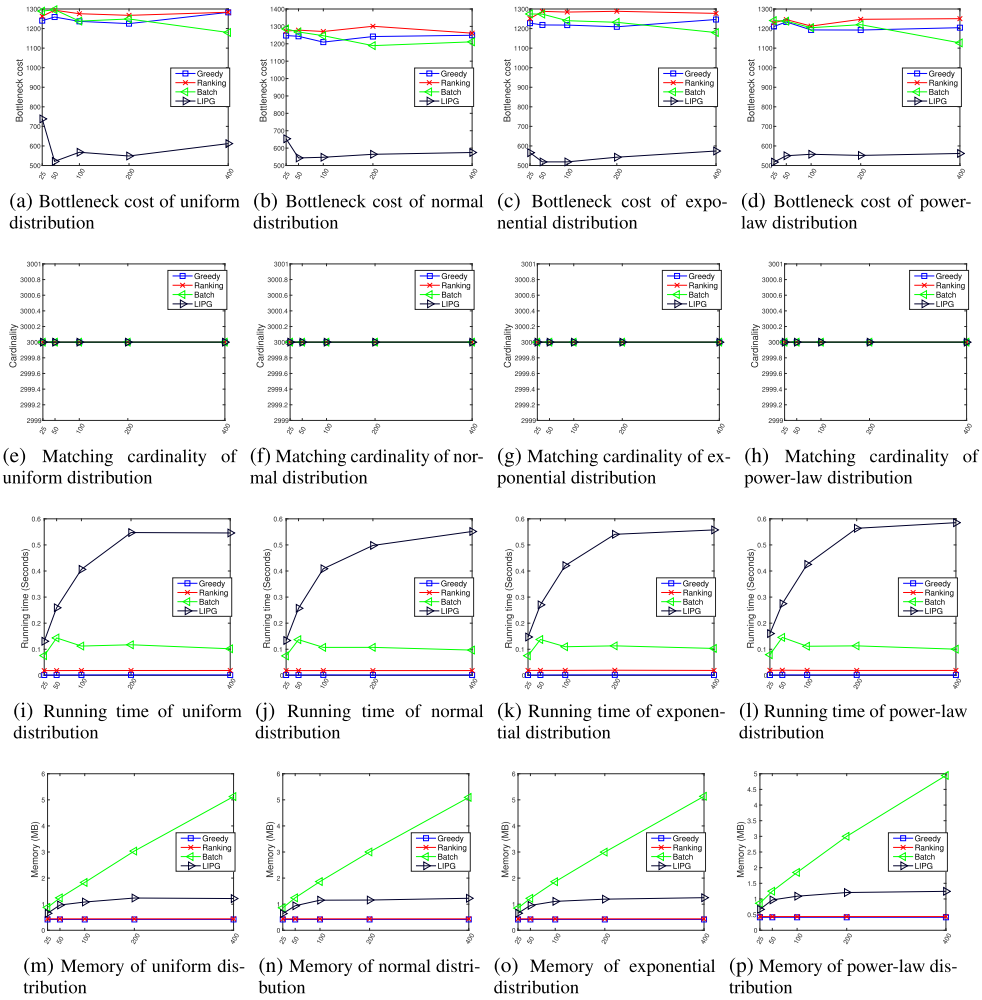
where $T' = \{\forall t \in T | dis(t, w_j) \leq \varphi\}$.

For a task t_i, the TLD of t_i is defined as the number of workers surrounding around it within the distance range of φ. If there are less workers surround t_i, t_i will be more isolated. And TLD of a worker is defined almost the same. Hence, TLD measures how isolated a task/worker is.

Notice that different settings of the threshold φ can significantly affect the result of TLD. In this paper, we set $\varphi = \eta \times \delta$, where δ is the average distance of arbitrary task-worker pair and η is the scaling factor. It turns out this setting $\varphi = \eta \times \delta$ can adaptively fit different spatial distribution setting of tasks and workers in our experiments. And η can also be empirically determined by experiments.

Based on the aforementioned discussion, we propose an online algorithm to settle the FOBMD problem, Local Isolated Point Greedy (LIPG), and the procedure of LIPG is shown in Alg. 3.

The procedure of LIPG is illustrated in Alg. 3. In line 2, we add the new arrival task t to the task set T. In line 4, we add the new arrival worker w to the worker set W. Lines 6-36 are the procedure of matching. We first update the task set T and worker set W by deleting the already



(a) Bottleneck cost of uniform distribution (b) Bottleneck cost of normal distribution (c) Bottleneck cost of exponential distribution (d) Bottleneck cost of power-law distribution

(e) Matching cardinality of uniform distribution (f) Matching cardinality of normal distribution (g) Matching cardinality of exponential distribution (h) Matching cardinality of power-law distribution

(i) Running time of uniform distribution (j) Running time of normal distribution (k) Running time of exponential distribution (l) Running time of power-law distribution

(m) Memory of uniform distribution (n) Memory of normal distribution (o) Memory of exponential distribution (p) Memory of power-law distribution

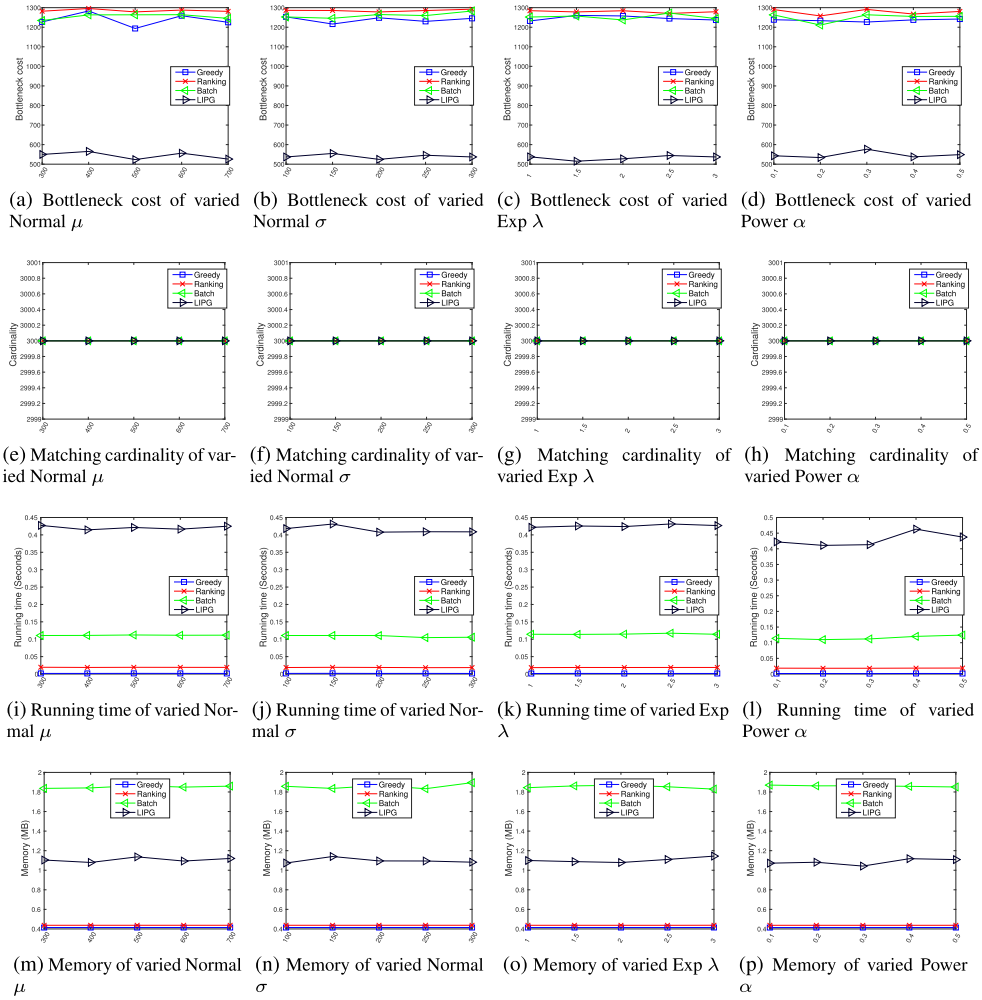
FIGURE 6. Results that the locations of task/workers follow uniform, normal, exponential and power-law distributions with different deadline settings.

expired tasks/workers in line 6. In lines 7-8, we calculate the average distance δ of all possible task-worker matching pairs. In lines 9-10, we initialize TLD of all tasks and workers as 0. In lines 11-13, we calculate the TLD for each task. In lines 14-16, we calculate the TLD for each worker. And in lines 17-25, we match all the tasks reaching their own deadlines. If the task's TLD is 0, we greedily match the task to its nearest unmatched worker, otherwise, we greedily match the task to the worker with largest TLD within the distance range of $\kappa \times \delta$ in lines 19-24. After matching each time, we delete the already matched worker from W in line 25. And we update the task set T by deleting the already matched tasks in line 26. Similarly, we match the workers which reach their deadline in lines 27-35. And we update the worker set W in line 36.

Notice that there are two different thresholds as shown in Alg. 3, the threshold of computing the local density (line 12 and line 15) and the threshold used when matching (line 22 and 32). The former threshold decides the TLD of tasks and workers, and the latter threshold is designed to designed for prevent the distance cost of a task-worker pair from becoming too large. Different thresholds have different purposes and they should be different. As mentioned before, we set the threshold of computing the local density (line 12 and line 15) as $\eta \times \delta$ to make this threshold able to adaptive fit different distribution settings of tasks and workers. Similarly, we also set the latter threshold as $\kappa \times \delta$ to adaptively fit different distribution settings of tasks and workers.

Back to Example 2, we assume $\eta = 1$ and $\kappa = 1$. The fist task t_0 is matched at time 11. At time 11, $T = \{t_0, t_1, t_2, t_3, t_4, t_5\}$ and $W = \{w_0, w_1, w_2, w_3, w_4\}$. And $\delta = 279 + \epsilon$, $\delta = \frac{279+\epsilon}{6*5} \approx 9.3$, and $TLD(T) = \{4, 4, 4, 5, 2, 0\}$, $TLD(W) = \{4, 4, 4, 5, 2\}$. And t_0 will be matched to w_0. Similarly, t_1 will be matched to w_1 at time 12. Finally, $\{t_0, t_1, \ldots, t_{k-1}\}$ are matched to $\{w_0, w_1, \ldots, w_{k-1}\}$, respectively and the cardinality is k, the bottleneck cost is $1 + \epsilon$.

Complexity Analysis: The time and space complexity of LIPG for each matching procedure are $O(|T| \cdot |W|)$ and $O(|T| \cdot |W|)$, respectively. When $|T| = |W| = n$, the time complexity of LIPG is $O(n^2)$, which is polynomial, and



(a) Bottleneck cost of varied Normal μ (b) Bottleneck cost of varied Normal σ (c) Bottleneck cost of varied Exp λ (d) Bottleneck cost of varied Power α

(e) Matching cardinality of varied Normal μ (f) Matching cardinality of varied Normal σ (g) Matching cardinality of varied Exp λ (h) Matching cardinality of varied Power α

(i) Running time of varied Normal μ (j) Running time of varied Normal σ (k) Running time of varied Exp λ (l) Running time of varied Power α

(m) Memory of varied Normal μ (n) Memory of varied Normal σ (o) Memory of varied Exp λ (p) Memory of varied Power α

FIGURE 7. Results that the locations of workers follow normal, exponential and power-law distributions while the locations of tasks follow uniform distribution.

we can say that LIPG is theoretically efficient in running time.

Implementation of LIPG: Note that LIPG consumes most of running time for computing the distance between an arbitrary task and an arbitrary worker in Line 7 as shown in Alg. 3. And the distance computing in line 12, line 15, line 22 and line 32 is contained in line 7. To reduce the redundant computation of distance between tasks and workers, we use a dynamic distance matrix to cache the distance of all available tasks and workers. When a new task/worker arrives, we compute the distances between the task/worker and all available workers/tasks, and we insert these distances to the distance matrix. After matching a task and a worker, we delete all the distances corresponding the task and the worker from the distance matrix. It turns out that using the dynamic distance matrix can really make LIPG more efficient in running time.

V. EXPERIMENTAL EVALUATION

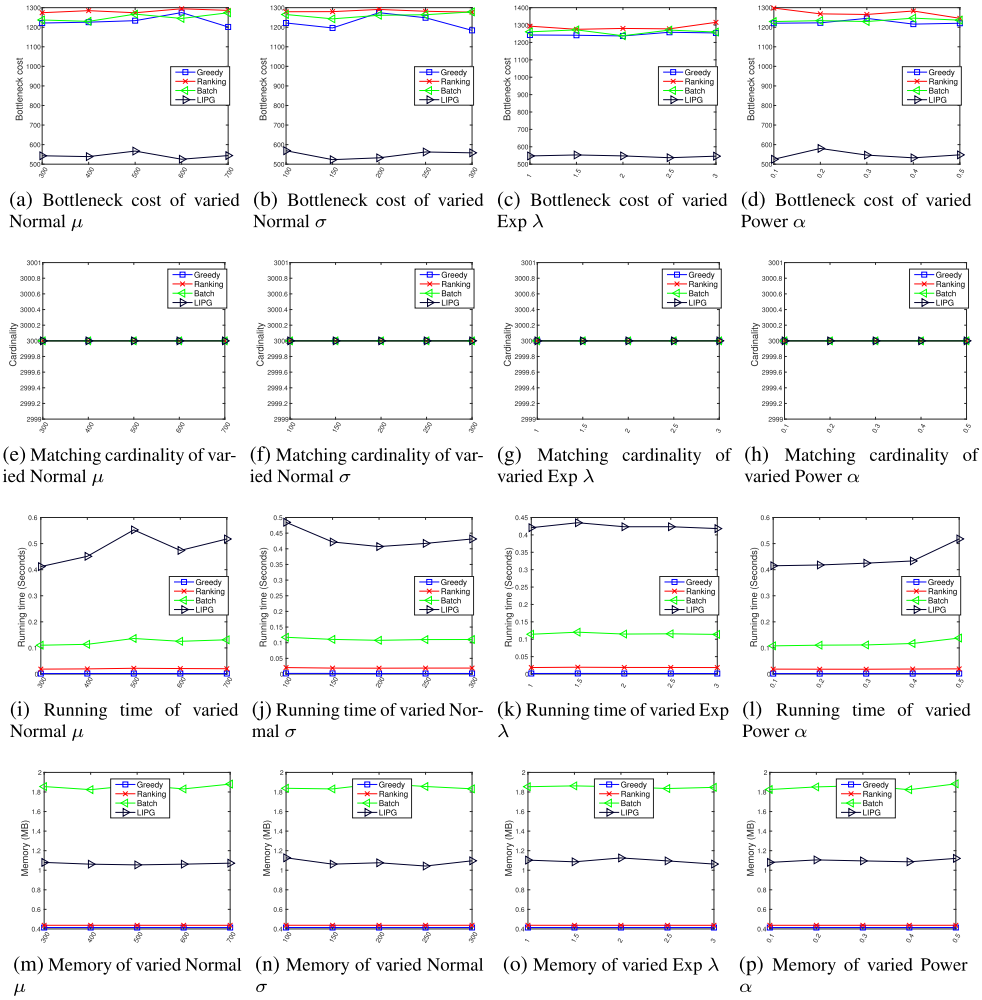
In this section, we conduct experiments on both synthetic and real datasets with all the baseline algorithms and our proposed algorithm, LIPG. Firstly, we introduce the experimental settings in Sec. V-A. And then we demonstrate the effectiveness and efficiency of our proposed algorithms in Sec. V-B.

A. EXPERIMENTAL SETTINGS

Synthetic Dataset: We use a 1000*1000 2D grid space as the working space and the Euclidean distance function as the distance metric $dis(\cdot, \cdot)$. Four different spatial distributions (uniform, normal, power-low and exponential distributions) are taken into consideration to validate the effectiveness and efficiency of the proposed algorithm. Notice that these spatial distributions have been generally used in studies about online matching problems in SC [5], [6], [9]–[11], [41], [42] and that is the reason why we choose these distributions.

For the arriving time, we assume all time and workers arrive at discrete time stamps randomly sampled from $\{0, 1, 2, \ldots, 1000\}$, *i.e.* in the time period $[0, 1000]$.

Specific parameters of different settings are shown in Table 1. The bold parameters in Table 1 is the default parameters in experiments.



(a) Bottleneck cost of varied Normal μ (b) Bottleneck cost of varied Normal σ (c) Bottleneck cost of varied Exp λ (d) Bottleneck cost of varied Power α

(e) Matching cardinality of varied Normal μ (f) Matching cardinality of varied Normal σ (g) Matching cardinality of varied Exp λ (h) Matching cardinality of varied Power α

(i) Running time of varied Normal μ (j) Running time of varied Normal σ (k) Running time of varied Exp λ (l) Running time of varied Power α

(m) Memory of varied Normal μ (n) Memory of varied Normal σ (o) Memory of varied Exp λ (p) Memory of varied Power α

FIGURE 8. Results that the locations of workers follow normal, exponential and power-law distributions while the locations of tasks follow normal distribution.

TABLE 1. Synthetic distributions' setting.

Name	Parameters
Uniform Distribution	None
Normal Distribution	μ=300, 400, **500**, 600, 700, σ =100, 150, **200**, 250, 300
Exponential Distribution	λ=1, 1.5, **2**, 2.5, 3
Power-law Distribution	α=0.1, 0.2, **0.3**, 0.4, 0.5
Scalability	$\|T\| = \|W\|$ =1000, 2000, **3000**, 4000, 5000
Deadline	25, 50, **100**, 200, 400

Real-World Dataset: We use the open datasets collected by DiDi Chuxing, the largest taxi dispatching platform in China. The dataset contains 7,063,532 online taxi-calling orders with corresponding pick-up/drop-out time and GPS locations of November 2016 in Chengdu, China. And we assume the pick-up locations and time as the locations and time of a task appearing on the platform. Similarly, we assume workers' locations and time are the drop-out locations and time. Hence, we have a task dataset with 7,063,532 tasks and a worker dataset with 7,063,532 workers.

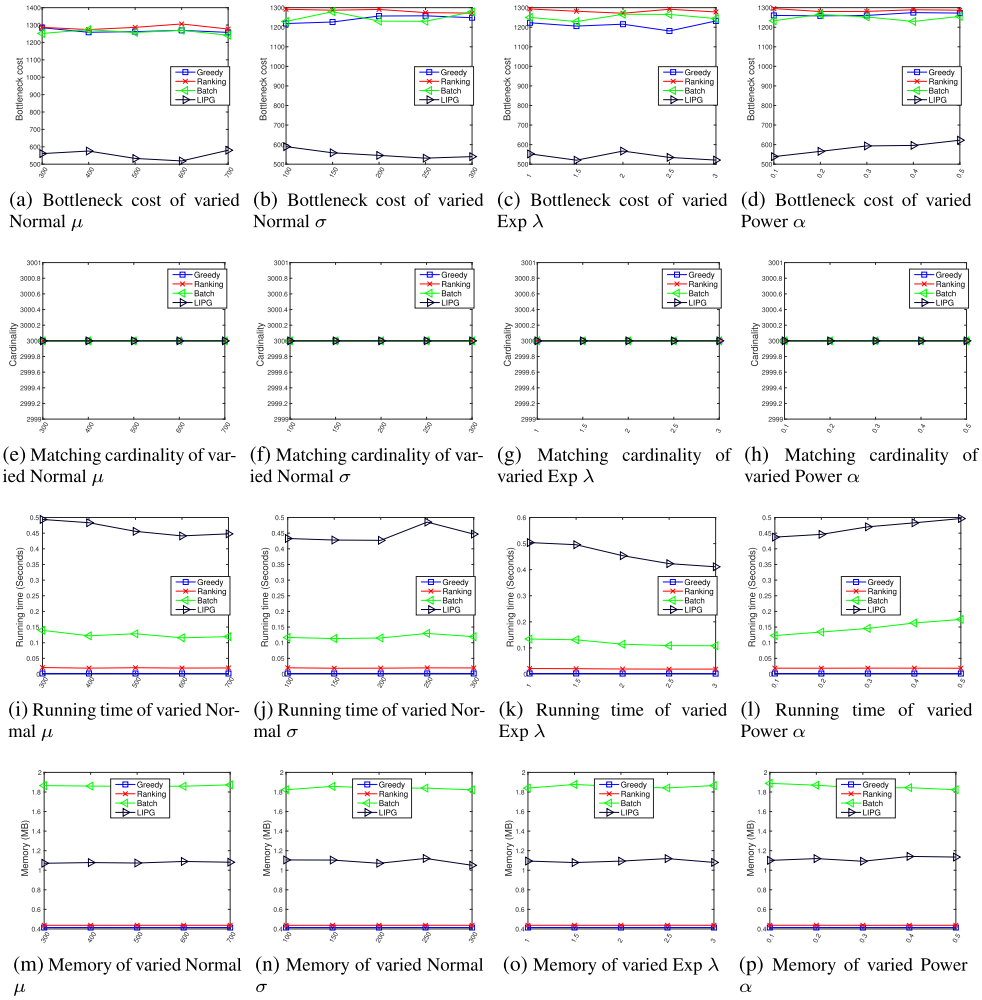
In our experiments, we randomly sample different numbers of tasks and workers in from all 7,063,532 tasks and workers, respectively. And the distance function based on GPS is used as the distance function $dis(\cdot, \cdot)$ (accurate into one kilometer). Notice that we normalize the arriving time of the sampled tasks/workers to [0, 1000] for avoiding the effect of different time unit settings (second or minute).

Evaluation Metrics and Implementation Details: We study the effect of varying parameters on the performance of different algorithms in terms of bottleneck cost, cardinality, running time and memory consuming. All the experiments' results are the average performance of these algorithms running repeatedly for 10 times.

All algorithms were implemented in C++ and the experiments were performed on a PC with Intel(R) Core(TM) i7-7700 CPU and 32G memory.

B. EXPERIMENT RESULTS

We first analyze the effect of different setting of Batch's θ and LIPG's η under different distributions. And then, we show



(a) Bottleneck cost of varied Normal μ (b) Bottleneck cost of varied Normal σ (c) Bottleneck cost of varied Exp λ (d) Bottleneck cost of varied Power α

(e) Matching cardinality of varied Normal μ (f) Matching cardinality of varied Normal σ (g) Matching cardinality of varied Exp λ (h) Matching cardinality of varied Power α

(i) Running time of varied Normal μ (j) Running time of varied Normal σ (k) Running time of varied Exp λ (l) Running time of varied Power α

(m) Memory of varied Normal μ (n) Memory of varied Normal σ (o) Memory of varied Exp λ (p) Memory of varied Power α

FIGURE 9. Results that the locations of workers follow normal, exponential and power-law distributions while the locations of tasks follow exponential distribution.

the performance of the aforementioned algorithms under different distributions and different scalability settings. Finally, we show the performance of these algorithms one the real dataset.

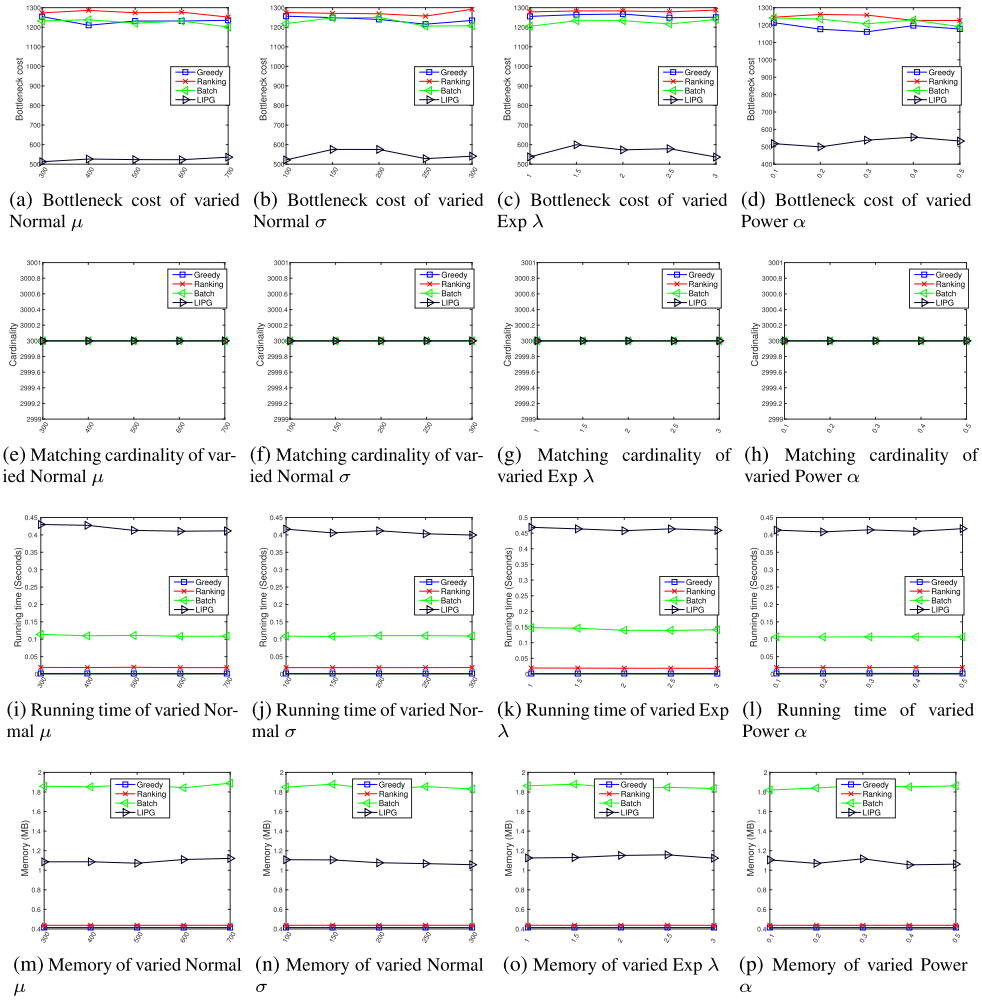
Effect of Batch's θ: Fig. 3 shows the effect of the different settings of θ in Batch when tasks/workers follow different spatial distributions with the default scalability settings of $|T| = |W| = 3000$. The settings of θ are $\{0.2, 0.4, 0.6, 0.8, 1\}\times$deadline. And the bottleneck cost becomes a bit lower (about 10%) when θ increases from $0.2\times$deadline to $1\times$deadline. The matching cardinality do not change with θ increasing, and Batch always achieve the maximum-cardinality matching. The running time increases and decreases with θ increasing. And the memory consumed by Batch increases with θ increase because number of tasks/workers increases in each batch.

Taking the bottleneck cost, running time and memory into consideration, we use $\theta = 1\times$deadline as the default setting of θ in next experiments.

Effect of LIPG's η: We choose $\eta = \{0.5, 1, 1.5, 2, 2.5\}$ in this experiment and the scalability is $|T| = |W| = 3000$. Fig. 4 shows the different settings of η does affect the bottleneck cost, running time and memory under different distributions. Fig. 4a shows the bottleneck cost decreases first and then increases with η increases. Fig. 4b shows LIPG always can output maximum-cardinality matching with different η. Fig. 4c and Fig. 4d show the running time and memory change dramatically when η increases for 2 to 2.5.

Based on aforementioned observation, we use $\eta = 2$ as the default setting in all following experiments including the experiments on the real dataset.

Effect of LIPG's κ: We choose $\kappa = \{0.5, 0.75, 1, 1.25, 1.5\}$ in this experiment with the scalability $|T| = |W| = 3000$. As shown in Fig. 5, different settings of κ affect the bottleneck cost, running time and memory under different distributions. Similarly with the effect of LIPG's η, Fig. 5a shows the bottleneck cost decreases first and then increases with κ increases. Note that the bottleneck is lowest when $\kappa = 1$ in



(a) Bottleneck cost of varied Normal μ (b) Bottleneck cost of varied Normal σ (c) Bottleneck cost of varied Exp λ (d) Bottleneck cost of varied Power α

(e) Matching cardinality of varied Normal μ (f) Matching cardinality of varied Normal σ (g) Matching cardinality of varied Exp λ (h) Matching cardinality of varied Power α

(i) Running time of varied Normal μ (j) Running time of varied Normal σ (k) Running time of varied Exp λ (l) Running time of varied Power α

(m) Memory of varied Normal μ (n) Memory of varied Normal σ (o) Memory of varied Exp λ (p) Memory of varied Power α

FIGURE 10. Results that the locations of workers follow Normal, Exponential and Power-law distributions while the locations of tasks follow power-law distribution.

most cases, but the bottleneck reaches the lowest value when $\eta = 2$ in most cases. And we can say that κ and η should be different as shown in Fig. 5a and Fig. 4a. And the two threshold, $\kappa \times \delta$ and $\eta \times \delta$, are different.

Fig. 5b shows LIPG always can output maximum-cardinality matching with different κ. Fig. 5c and Fig. 5d show the running time and memory decrease when κ increases for 0.5 to 1.5.

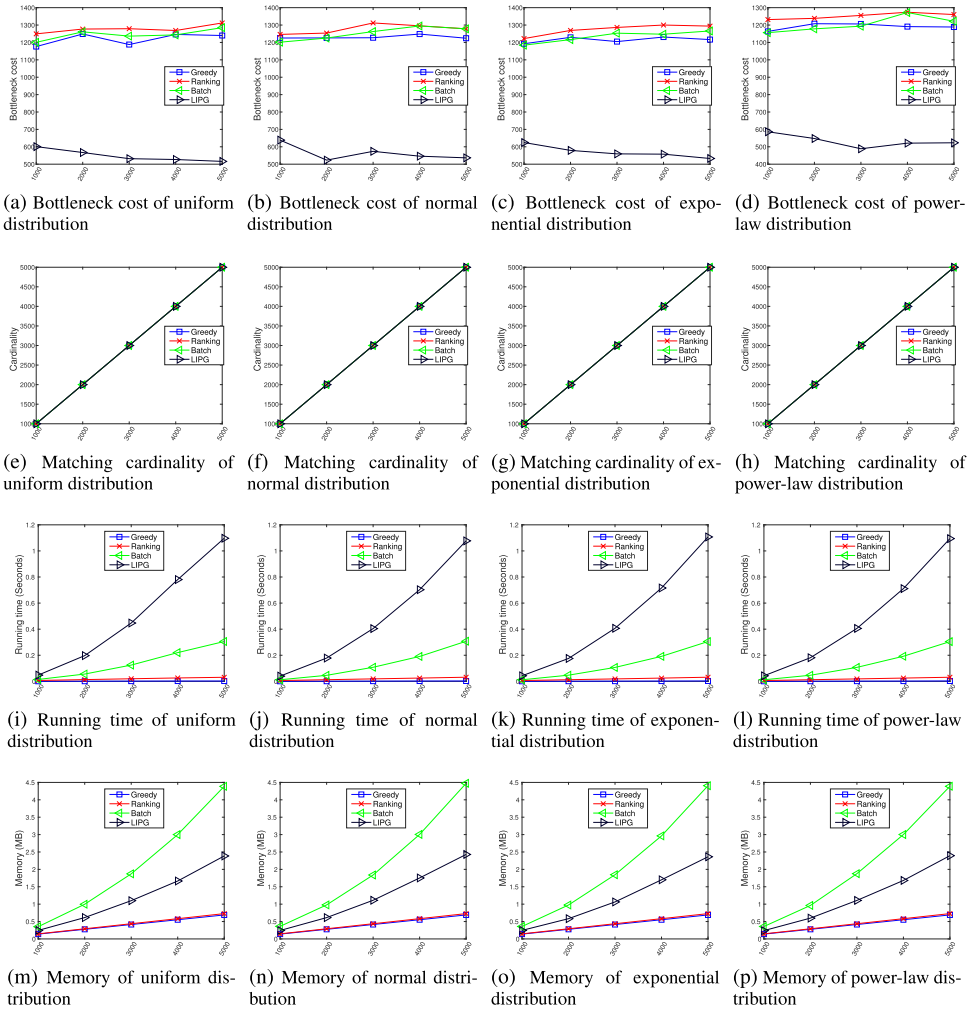
Based on aforementioned observation, we choose $\kappa = 1$ as the default setting in all following experiments including the experiments on the real dataset.

Effect of Deadline: Fig. 6 shows the bottleneck cost remains nearly stable under different deadlines ($\{25, 50, 100, 200, 400\}$). And the matching cardinality is not affected by different deadlines.

The running time of LIPG increases with deadlines increase and the running time of other algorithms remains nearly stable. And the memory of Batch increases with deadlines increase, but the memory of other algorithms remains stable with deadlines increase.

Generally, the different settings of deadlines do not affect the matching result. And we set the deadlines of tasks/workers as 100 in the following experiments.

Effect of Locations of Tasks Following Different Distribution: Fig. 7 shows the results when the locations of tasks follow the uniform distribution and the locations of workers follow three different distributions respectively, normal, exponential and power-law distributions. Fig. 8 shows the results when the locations of tasks follow normal distribution and the locations of workers follow normal, exponential and power-law distributions with different parameters, respectively. Fig. 9 shows the results when the locations of tasks follow exponential distribution and the locations of workers follow normal, exponential and power-law distributions with different parameters, respectively. And Fig. 10 shows the results when the locations of tasks follow power-law distribution and the locations of workers follow normal, exponential and power-law distributions with different parameters, respectively. Notice that the scalability is $|T| = |W| = 3000$ in these experiments.



(a) Bottleneck cost of uniform distribution (b) Bottleneck cost of normal distribution (c) Bottleneck cost of exponential distribution (d) Bottleneck cost of power-law distribution

(e) Matching cardinality of uniform distribution (f) Matching cardinality of normal distribution (g) Matching cardinality of exponential distribution (h) Matching cardinality of power-law distribution

(i) Running time of uniform distribution (j) Running time of normal distribution (k) Running time of exponential distribution (l) Running time of power-law distribution

(m) Memory of uniform distribution (n) Memory of normal distribution (o) Memory of exponential distribution (p) Memory of power-law distribution

FIGURE 11. Results that the locations of workers follow uniform, normal, exponential and power-law distributions with different scalability settings.

For the bottleneck cost, we observe that LIPG always outperforms other algorithms and the bottleneck cost of LIPG is only about 50% of the bottleneck cost of other algorithms. And the bottleneck cost of Greedy, Ranking and Batch is similar. In most cases, Ranking outputs the bottleneck cost a bit larger than the bottleneck cost of Greedy and Batch because Ranking treats all tasks and workers equally and there is no strategy in Ranking to resist the sensibility of bottleneck cost.

For matching cardinality, all algorithms can achieve the maximum-cardinality matching because all algorithms do not actively refuse a task or a worker.

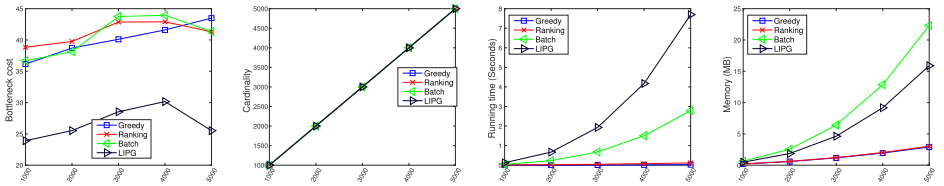
As for the running time, LIPG is worst comparing to other algorithms, but LIPG is still efficient enough in running time because LIPG takes only about 0.5 seconds for matching 3000 tasks and workers. And Batch takes more time then Greedy and Ranking because the calculation of local optimal bottleneck matching in each batch. Greedy runs fastest than other algorithms and Ranking runs a bit slower than Greedy because Ranking has to pick a random float value for each task/worker.

When considering memory consuming, Batch consumes most memory then other algorithms because the local optimal bottleneck matching calculation in each batch. Greedy, Ranking and LIPG are as efficient as the same.

Scalability: We study the effect of scalability for the algorithms, where the size of $|T|$ ($|W|$) varied from 1000 to 5000. And Fig. 11 shows the results of experiments of the scalability with tasks/workers follow uniform, normal, exponential and power-law distributions. All these figures show that LIPG always outperforms other online algorithms at the bottleneck cost under all situations. The bottleneck cost of LIPG is about half of the bottleneck cost of other algorithms. And Greedy, Ranking and Batch output similar bottleneck cost. In most cases, Greedy's bottleneck cost is a bit lower than Ranking and Batch, and Ranking's bottleneck cost is worst among all four algorithms.

As for the matching cardinality, all four algorithms can output the maximum cardinality.

The running time of all four algorithms in descending order is LIPG>Batch>Ranking>Greedy. LIPG takes less than 1.2 seconds for LIPG to match 5000 tasks and workers.



(a) Bottleneck cost of the real dataset (b) Running time of the real dataset (c) Running time of the real dataset (d) Memory of the real dataset

FIGURE 12. Results on the real dataset.

And LIPG is still quite fast and is still experimentally efficient in running time.

The memory of all four algorithms in descending order is Batch >LIPG≈Ranking≈Greedy. Batch costs much more memory than other algorithms due to the calculating process of the local optimal bottleneck matching in each batch.

Experiments on the Real Dataset: Fig. 12 shows the results of the experiment on the real dataset with the size of $|T|$ ($|W|$) varies from 1000 to 5000. Obviously, the matching results of LIPG have much lower bottleneck cost than other algorithms as shown in Fig. 12a. The bottleneck cost of LIPG remains relatively stable and does not increases fast with the size of $|T|$ ($|W|$) increases from 1000 to 5000.

Fig. 12b shows all four algorithms can output the maximum-cardinality matching.

Fig. 12c shows that LIPG takes more running time than other algorithms and the running time of all four algorithms in descending order is LIPG>Batch>Ranking≈Greedy. LIPG only takes less than 8 seconds to match 5000 tasks and workers (0.0016 second to match a task and a worker), and the time complexity of the LIPG algorithm is polynomial, $O(n^2)$ when $|T| = |W| = n$. So, we can still say LIPG is acceptable and efficient, experimentally and theoretically. And Batch also takes about 2 seconds to match 5000 tasks and workers. Notice that LIPG and Batch consume more running-time on the real dataset then them on the synthetic datasets with the same number of tasks/workers, and the reason is that it is much more time-consuming for calculating the GPS distance than calculating the simple euclidean distance.

Fig. 12d shows that Batch consumes more memory than other algorithms. The consuming memory of four algorithms in descending order is Batch>LIPG>Ranking≈Greedy. And LIPG is efficient in memory.

Summary of Experiments: We summarize the observation of all aforementioned experiments as follows:

- LIPG always outputs the matching result with much lower bottleneck cost than Greedy, Ranking and Batch.
- Greedy, Ranking, Batch and LIPG can achieve the maximum-cardinality matching.
- LIPG consumes more time than Greedy, Ranking and Batch, but still is efficient in running time.
- LIPG is efficient in memory and consumes much less memory than Batch.

VI. CONCLUSION

In this paper, we formulate the Fully Online Bottleneck Matching with Deadlines (FOBMD) problem and clarify that there is no online algorithm without actively refusing tasks can achieve a constant competitive ratio of the bottleneck cost for the FOBMD problem. And then we introduce three baseline algorithms, Greedy, Ranking and Batch. After that, we discuss how to lower the bottleneck cost by considering the isolated tasks/workers, and we propose the Local Isolated Point Greedy (LIPG) algorithm to handle the FOBMD problem based on the discussion. Finally, extensive experiments on both synthetic and real datasets demonstrate that LIPG can output the maximum-cardinality matching with the drastically lower bottleneck cost than the other online algorithms and LIPG is also efficient in running time and memory.

REFERENCES

[1] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi, "Spatial crowdsourcing: A survey," *VLDB J.*, vol. 29, no. 1, pp. 217–250, Jan. 2019.
[2] Y. Tong, L. Chen, and C. Shahabi, "Spatial crowdsourcing: Challenges, techniques, and applications," *Proc. VLDB Endowment*, vol. 10, no. 12, pp. 1988–1991, Aug. 2017.
[3] L. Kazemi, C. Shahabi, and L. Chen, "Geotrucrowd: Trustworthy query answering with spatial crowdsourcing," in *Proc. GIS*, 2013, pp. 314–323.
[4] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu, "Flexible online task assignment in real-time spatial data," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1334–1345, Aug. 2017.
[5] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *Proc. ICDE*, May 2016, pp. 49–60.
[6] T. Song, Y. Tong, L. Wang, J. She, B. Yao, L. Chen, and K. Xu, "Trichromatic online matching in real-time spatial crowdsourcing," in *Proc. ICDE*, Apr. 2017, pp. 1009–1020.
[7] H. To, C. Shahabi, and L. Kazemi, "A server-assigned spatial crowdsourcing framework," *ACM Trans. Spatial Algorithms Syst.*, vol. 1, no. 1, pp. 1–28, Jul. 2015.
[8] Q. Tao, Y. Zeng, Z. Zhou, Y. Tong, L. Chen, and K. Xu, "Multi-worker-aware task planning in real-time spatial crowdsourcing," in *Proc. DASFAA*, 2018, pp. 301–317.
[9] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1053–1064, 2016.
[10] L. Li, J. Fang, B. Du, and W. Lv, "Spatial bottleneck minimum task assignment with time-delay," in *Proc. DASFAA*, 2019, pp. 387–391.
[11] Z. Chen, P. Cheng, Y. Zeng, and L. Chen, "Minimizing maximum delay of task assignment in spatial crowdsourcing," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1454–1465.
[12] Y. Zeng, Y. Tong, L. Chen, and Z. Zhou, "Latency-oriented task completion via spatial crowdsourcing," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 317–328.
[13] U. U. Hassan and E. Curry, "A multi-armed bandit approach to online spatial task assignment," in *Proc. IEEE UIC*, Dec. 2014, pp. 212–219.
[14] Y. Li, J. Fang, Y. Zeng, B. Maag, Y. Tong, and L. Zhang, "Two-sided online bipartite matching in spatial data: Experiments and analysis," *GeoInformatica*, vol. 24, no. 1, pp. 175–198, Jan. 2019.

[15] B. Zhao, P. Xu, Y. Shi, Y. Tong, Z. Zhou, and Y. Zeng, "Preference-aware task assignment in on-demand taxi dispatching: An online stable matching approach," in *Proc. AAAI*, 2019, pp. 2245–2252.
[16] Y. Tong, Y. Zeng, B. Ding, L. Wang, and L. Chen, "Two-sided online micro-task assignment in spatial crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, early access, Nov. 13, 2019, doi: 10.1109/TKDE.2019.2948863.
[17] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *Proc. VLDB Endowment*, vol. 11, no. 11, pp. 1633–1646, Jul. 2018.
[18] Y. Zeng, Y. Tong, and L. Chen, "Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees," *Proc. VLDB Endowment*, vol. 13, no. 3, pp. 320–333, 2019.
[19] D. Gao, Y. Tong, J. She, T. Song, L. Chen, and K. Xu, "Top-k team recommendation and its variants in spatial crowdsourcing," *Data Sci. Eng.*, vol. 2, no. 2, pp. 136–150, Jun. 2017.
[20] Z. Huang, N. Kang, Z. G. Tang, X. Wu, Y. Zhang, and X. Zhu, "How to match when all vertices arrive online," in *Proc. 50th Annu. ACM SIGACT Symp. Theory Comput. (STOC)*, 2018, pp. 17–29.
[21] Z. Huang, B. Peng, Z. G. Tang, R. Tao, X. Wu, and Y. Zhang, "Tight competitive ratios of classic matching algorithms in the fully online model," in *Proc. SODA*. Philadelphia, PA, USA: SIAM, 2019, pp. 2875–2886.
[22] I. Ashlagi, M. Burq, C. Dutta, P. Jaillet, A. Saberi, and C. Sholley, "Edge weighted online windowed matching," in *Proc. ACM EC*, 2019, pp. 729–742.
[23] I. Ashlagi, M. Burq, P. Jaillet, and A. Saberi, "Maximizing efficiency in dynamic matching markets," 2018, *arXiv:1803.01285*. [Online]. Available: http://arxiv.org/abs/1803.01285
[24] D. Fulkerson, *A Production Line Assignment Problem*. Santa Monica, CA, USA: RAND Corporation, 1953.
[25] H. N. Gabow and R. E. Tarjan, "Algorithms for two bottleneck optimization problems," *J. Algorithms*, vol. 9, no. 3, pp. 411–417, Sep. 1988.
[26] R. S. Garfinkel, "An improved algorithm for the bottleneck assignment problem," *Oper. Res.*, vol. 19, no. 7, pp. 1747–1751, 1971.
[27] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*, vol. 106. Philadelphia, PA, USA: SIAM, 2012.
[28] U. Derigs and U. Zimmermann, "An augmenting path method for solving linear bottleneck assignment problems," *Computing*, vol. 19, no. 4, pp. 285–295, Dec. 1978.
[29] O. Gross, *The Bottleneck Assignment Problem*. Santa Monica, CA, USA: RAND Corporation, 1959.
[30] C. Long, R. C.-W. Wong, P. S. Yu, and M. Jiang, "On optimal worst-case matching," in *Proc. ACM SIGMOD*, 2013, pp. 845–856.
[31] R. Idury and A. Schaffer, "A better lower bound for on-line bottleneck matching, manuscript," Dept. Comput. Sci., Rice Univ., Houston, Texas., 1992. [Online]. Available: http://www.ncbi.nlm.nih.gov/core/assets/cbb/files/Firehouse.pdf
[32] B. M. Anthony and C. Chung, "Online bottleneck matching," *J. Combinat. Optim.*, vol. 27, no. 1, pp. 100–114, 2014.
[33] B. M. Anthony and C Chung, "The power of rejection in online bottleneck matching," in *Proc. 8th Annu. Int. Conf. Combinat. Optim. Appl. (COCOA)* (Lecture Notes in Computer Science). Springer, 2014, pp. 395–411.
[34] B. M. Anthony and C. Chung, "Serve or skip: The power of rejection in online bottleneck matching," *J. Combinat. Optim.*, vol. 32, no. 4, pp. 1232–1253, Nov. 2016.
[35] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, "An optimal algorithm for on-line bipartite matching," in *Proc. ACM STOC*, 1990, pp. 352–358.
[36] B. Kalyanasundaram and K. Pruhs, "Online weighted matching," *J. Algorithms*, vol. 14, no. 3, pp. 478–488, May 1993.
[37] N. R. Devanur, K. Jain, and R. D. Kleinberg, "Randomized primal-dual analysis of RANKING for online bipartite matching," in *Proc. SODA*. Philadelphia, PA, USA: SIAM, Jan. 2013, pp. 101–107.
[38] L. Kazemi and C. Shahabi, "Geocrowd: Enabling query answering with spatial crowdsourcing," in *Proc. ACM GIS*, 2012, pp. 189–198.
[39] A. V. Goldberg, S. Hed, H. Kaplan, R. E. Tarjan, and R. F. Werneck, "Maximum flows by incremental breadth-first search," in *Proc. ESA*. Berlin, Germany: Springer, 2011, pp. 457–468.
[40] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. SIGMOD*, 2000, vol. 29, no. 2, pp. 93–104.
[41] Z.-Q. Jiang, W.-J. Xie, M.-X. Li, B. Podobnik, W.-X. Zhou, and H. E. Stanley, "Calling patterns in human communication dynamics," *Proc. Nat. Acad. Sci. USA*, vol. 110, no. 5, pp. 1600–1605, Jan. 2013.
[42] X. Liang, J. Zhao, L. Dong, and K. Xu, "Unraveling the origin of exponential law in intra-urban human mobility," *Sci. Rep.*, vol. 3, no. 1, Dec. 2013, Art. no. 2983.

LONG LI is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Beihang University. His major research interests include crowdsourcing and spatio-temporal data management.

WEIFENG LV received the B.S. degree in computer science and engineering from Shandong University, Jinan, China, in 1992, and the Ph.D. degree in computer science and engineering from Beihang University, Beijing, China, in 1998. He is currently a Professor with the State Key Laboratory of Software Development Environment, Beihang University. His research interests include smart city technology and mass data processing.