**SCIENCE CHINA**
Information Sciences

• **Supplementary File** •

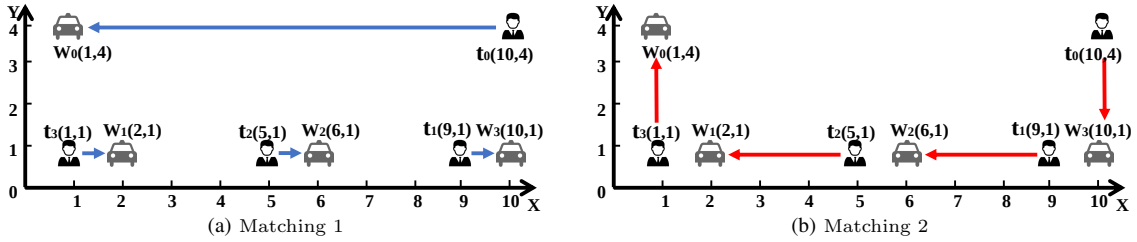
# Real-Time Bottleneck Matching in Spatial Crowdsourcing

Long LI[1], Lingling WANG[2*] & Weifeng LV[1]

[1]*SKLSDE Lab and BDBC, Beihang University, Beijing* 100191, *China;*
[2]*Beijing Institute of Technology, Beijing* 100081, *China*

## Appendix A Reason Why We Study The Bottleneck Optimization Objective

The exist researches about the real-time matching problem in SC often focus on minimizing the total distance cost of all worker-task matching pairs [11, 19–22]. Such optimization objective only considers the overall distance cost and the distance of a few worker-task matching pairs can be unusually large by comparing with others under such optimization objective. These unusually large distances weaken the users' experience. For example, the unusually large distance means extraordinarily long waiting time before the passenger get picked up by the corresponding taxi driver in the taxi-calling platform. And extraordinarily long waiting time exhausts passengers' patience and makes them turn to another platform for a better service, which is the worst thing that the taxi-calling company wants to see. In this case, another optimization objective, minimizing the maximum distance cost among all worker-task matching pairs' distance cost, namely minimizing the bottleneck cost, can ensure the longest waiting time of all passengers is as short as possible and the users' experience is guaranteed. Hence, we study the bottleneck optimization objective in our paper, and an example to illustrate why we should study such an optimization objective is given as follows.




(a) Matching 1 (b) Matching 2

**Figure A1** An illustration of Example 1.

**Example 1.** Assume there are 4 taxi-calling tasks $\{t_0, t_1, t_2, t_3\}$ and 4 workers (drivers) $\{w_0, w_1, w_2, w_3\}$ in a 2D space as shown in Fig. A1. Assume the travel distance cost between tasks and workers are the Euclidean distance. There are two different minimum total cost matching results as shown in Fig. A1, and the total distance cost of both matching results is the same, 12. However, the bottleneck cost of these two matching results is tremendously different. The bottleneck cost of the matching 1 and matching 2 is 9 and 3, respectively. The unusually large distance cost in the matching 1, $distance(t_0, w_0) = 9$, means the corresponding user who releases task $t_0$ has to wait extremely long before getting picked up by $w_0$. And the user releasing $t_0$ can be extremely unsatisfied for the taxi-calling service. By contract, the matching 2 treats all tasks more equally with the bottleneck cost $distance(t_3, w_0) = 3$, and the lower bottleneck cost guarantees the users' experience.

## Appendix B Core Idea of LLDF

**Core Idea.** We observe some instances of the RMBM problem and find out that the spatial outliers pull the bottleneck cost to a very high level in many cases. Consider there are 4 tasks $\{t_0, t_1, t_2, t_3\}$ appearing one by one and 4 workers

* Corresponding author (email: linglingwang12@126.com)

$\{w_0, w_1, w_2, w_3\}$, and the locations of all tasks and workers are illustrated in Fig. B1. As shown in Fig. B1, $t_3$ is a task outlier and $w_0$ is a worker outlier. Both $t_3$ and $w_0$ have higher probability of causing the large bottleneck cost, especially when $t_3$ appears at last and all workers closer to $t_3$ are already occupied by other tasks. Notice that the spatial information of all tasks remains unknown until appearing, and then we cannot determine whether a task is an outlier or not. So, the task outliers are hard to handle. Fortunately, we can manage to handle the worker outliers. As shown in Fig. B1, the worker outlier $w_0$ locates far away from other workers. It is easy to ignore $w_0$ or leave $w_0$ at last and match $w_0$ to a further task instead of the optimal task $t_0$. In fact, we know all workers' spatial locations in advance and we can identify which workers are outliers. And then, we give the worker outliers the high priority to match when a proper task arriving. Taking Fig. B1 as an example, we can identify $w_0$ is an outlier and preferentially match $w_0$ to $t_0$ when $t_0$ arriving. Therefore, the bottleneck cost can be lowered. In a word, preferentially matching the worker outliers with the arriving tasks is our core idea to lower the bottleneck cost.

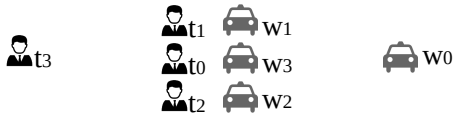



**Figure B1** An illustration for the task outlier and worker outlier.

Note that there are a lot of algorithms that can detect the worker outliers, such as LoF [2], kNN [17] and FastABOD [16] and so on. Considering the time efficiency for real-time matching, we use a simple metric, density, rather than complex outlier detection algorithms to determine whether a worker is an outlier or not. We assume the worker with lower density is more likely to be an outlier and should be matched first and the density of a worker is defined in Definition 2.

## Appendix C Competitive Ratio in The Adversarial Model

The Competitive Ratio (CR) is the standard theoretical evaluation for online algorithms and it is the ratio of the result of an online algorithm to the optimal result in the offline scenario where full information is known in advance. And the competitive ratio analysis always considers the worst-case ratio over all possible inputs and arrival orders.

**Definition 1** (Competitive Ratio)**.** The competitive ratio in the adversarial model of an online algorithm for the RMBM problem is shown as below:

$$CR_A = \max_{\forall T \ and \ \forall W \ and \ \forall \ \sigma \ of \ T} \frac{Cost(M)}{Cost(OPT)}$$

where $T/W$ is an arbitrary task/worker set, $\sigma$ is an arbitrary arrival order of $T$, $Cost(OPT)$ is the optimal bottleneck cost and $Cost(M)$ is the bottleneck cost of the matching generated by the online algorithm.

In RMBM, we can get $Cost(OPT)$ by offline bottleneck bipartite matching algorithms [5,6].

## Appendix D Proof of Theorem 1

Consider a case which all workers and tasks locate on a line, and $k$ workers, $\{w_0, w_1, w_2, ..., w_{k-1}\}$, locate at points $\{\frac{1}{\eta}, \frac{2}{\eta}, \frac{2^2}{\eta}, ..., \frac{2^{k-1}}{\eta}\}$ on the line respectively. Moreover, $k$ tasks, $\{t_0, t_1, t_2, ..., t_{k-1}\}$, appear at points $\{0, \frac{2}{\eta}, \frac{2^2}{\eta}, ..., \frac{2^{k-1}}{\eta}\}$ one by on the same line, respectively. Apparently, the optimal bottleneck matching algorithm would match $t_i$ to the corresponding $w_i$, for $i = 0, 1, ..., k-1$, and we have the optimal bottleneck cost is $Cost(OPT) = \frac{1}{\eta}$.

We can find that the distance $dis(w_{i-1}, w_i)$ is getting larger while $i$ increasing, and hence, the density of $w_i$ is getting lower with $i$ increasing. And when the task $t_0$ appears, LLDF matches $t_0$ to the worker $w_x$ with the lowest density within the range of $\eta \times Avg_{t_0}$, where $0 < \eta \leqslant 1$, $Avg_{t_0} = \frac{\sum_{j=0}^{j=|W|} dis(t_0, w_j)}{|W|}$. Apparently, $dis(t_0, w_x)$ is the finally bottleneck cost output by LLDF. We can conclude that $Cost(LLDF) = dis(t_0, w_x) = \frac{2^{\lfloor \log(\eta \cdot Avg_{t_0}) \rfloor}}{\eta}$, Where $Avg_{t_0} = \frac{\frac{2^{k-1}}{\eta} + \frac{2^{k-2}}{\eta} + ... + \frac{1}{\eta}}{k} = \frac{2^k - 1}{k \cdot \eta}$. And the optimal bottleneck cost in the offline scenario is $Cost(OPT) = \frac{1}{\eta}$.

So, we have $CR_{LLDF} = \frac{Cost(LLDF)}{Cost(OPT)} = \eta \cdot 2^{\lfloor \log(\frac{2^k - 1}{k}) \rfloor} = \eta \cdot 2^{\lfloor \log(2^k - 1) - \log k \rfloor} \geqslant \eta \cdot 2^{\lfloor k - \log k - 1 \rfloor}$.

Hence, LLDF's Competitive Ratio (CR) in the adversarial model is at least $\eta \cdot 2^{\lfloor k - \log k - 1 \rfloor}$, where $k = |M| = \min(|T|, |W|)$.

## Appendix E Experiments

### Appendix E.1 Baseline Algorithms Used in Experiments

#### Appendix E.1.1 *Greedy Algorithm*

Greedy [1, 9, 19] simply allocates each arrival task to its nearest unmatched worker.

**Complexity Analysis.** For each new task arriving at the SC platform, the time and space complexity of Greedy are $O(|W|)$.

**Competitive Analysis.** The competitive ratio of Greedy in the adversarial model is proven to be at least $2^{k-1}$, where $k = |M| = min(|T|, |W|)$ [1].

### Appendix E.1.2 *Permutation Algorithm*

Permutation [1, 9, 12] assigns tasks to workers as follows. After $t_i$ appearing, we consider the optimal bottleneck matching $M_1$ of $\{t_0, t_1, ..., t_i\}$ and $W$, and the optimal bottleneck matching $M_2$ of $\{t_0, t_1, ..., t_{i-1}\}$ and $W$. $M_1$ has exactly one more worker $w_x$ than $M_2$. And then, Permutation matches that worker $w_x$ to the current task $t_i$.

**Complexity Analysis.** Comparing with Greedy, Permutation is more complex and consumes more time. For each task arriving, Permutation's space complexity is $O(|W| \cdot |T|)$ and Permutation's time complexity is $O(\Omega \cdot log(max(|W|, |T|)))$ where $\Omega$ is the time complexity of a maximum flow algorithm (*e.g.* $\Omega = O(|W| \cdot |T| \cdot (|W| + |T|)^2)$) for using the IBFS algorithm [7].

**Competitive Analysis.** The Permutation's CR in the adversarial model is proven to be $2k-1$ ($k = |M| = min(|T|, |W|)$) [1].

### Appendix E.1.3 *Balance Algorithm*

Balance [1,10] is designed for the setting of multiple workers per task. We modify the original Balance for fitting the RMBM problem and we still name the modified Balance algorithm as "Balance".

When a new task $t_i$ arrives, Balance chooses the $t_i$'s nearest unmatched worker $w_x$ and the $t_i$'s second nearest unmatched worker $w_y$. If $dis(t_i, w_x)$ is smaller than $c \cdot dis(t_i, w_y)$ for a constant $0 < c < 1$, Balance assigns $t_i$ to $w_x$, or otherwise Balance assigns $t_i$ to $w_y$.

**Complexity Analysis.** For each new task, Balance's space and time complexity are $O(|W|)$.

**Competitive Analysis.** [9] has indicated that the worst case of Greedy for competitive analysis in the adversary model is when all the workers and tasks lie on a line. We follow this idea and give the competitive ratio analysis of Balance by reviewing a worst case.

**Theorem 1.** Balance's CR in the adversarial model is at least $2^{k-1}$, where $k = |M| = \min(|T|, |W|)$.

*Proof.* Consider all workers and tasks locating on a line, and $k$ workers, $\{w_0, ..., w_{k-1}\}$, locate at points $\{\varepsilon, \frac{2}{c}, \frac{2^2}{c}, ..., \frac{2^{k-1}}{c}\}$ on the line, respectively. Note that $\varepsilon$ is an arbitrarily small positive number. Moreover, $k$ tasks, $\{t_0, ..., t_{k-1}\}$, appear at points $\{\frac{1}{c}, \frac{2}{c}, \frac{2^2}{c}, ..., \frac{2^{k-1}}{c}\}$ one-by-one on the same line, respectively. Apparently, the optimal bottleneck matching algorithm would match $t_i$ to the corresponding $w_i$, for $i = 0, 1, ..., k-1$. And we have $Cost(OPT) = \frac{1}{c} - \varepsilon$.

When the task $t_0$ appears, $t_0$'s nearest unmatched worker is $w_1$ and the second nearest unmatched worker is $w_0$. And, $dis(t_0, w_0) = \frac{1}{c} - \varepsilon > c \cdot dis(t_0, w_1) = c \cdot \frac{1}{c} = 1$, where $0 < c < 1$. So, $t_0$ is matched to $w_1$ by Balance. Similarly, $t_i$ is matched to $w_{i+1}$ for $i = \{1, 2, ..., k-2\}$. When the task $t_{k-1}$ appears, there is only one worker left, $w_0$, and Balance has no choice but to match $t_{k-1}$ to $w_0$ with $dis(t_{k-1}, w_0) = \frac{2^{k-1}}{c} - \varepsilon$. Therefore, $Cost(Balance) = \frac{2^{k-1}}{c} - \varepsilon$. And we have $CR_{Balance} = \frac{Cost(Balance)}{Cost(OPT)} = \frac{\frac{2^{k-1}}{c} - \varepsilon}{\frac{1}{c} - \varepsilon} \approx 2^{k-1}$.

So, we can know that Balance's CR in the adversarial model is at least $2^{k-1}$.

### Appendix E.1.4 *Greedy-HST Algorithm.*

Greedy-HST [15, 20] is one of the state-of-the-art algorithm with theoretical guarantee for solving the online minimum matching problem in real-time SC. Initially, Greedy-HST constructs a 2-HST (2-Hierarchically Separated Tree) for all known workers and projects the workers from a 2D metric to a tree space. And when a task $t_i$ appearing, Greedy-HST firstly finds $t_i$'s nearest worker $w_j$ in the original 2D metric space whether $w_j$ is matched or not. Then, Greedy-HST chooses the unmatched worker $w_k$ nearest to $w_j$ on the tree metric and $w_k$ could be $w_j$ if $w_j$ is unmatched. If there are multiple unmatched workers with the same and nearest distance away from $w_j$, Greedy-HST randomly chooses one as $w_k$. We recommend [4, 15, 20] for more details about Greedy-HST and the procedure of constructing the 2-HST.

**Complexity Analysis.** For constructing the 2-HST, the time and space complexity is $O(k^2)$ and $O(\lceil \log \Delta \rceil \cdot k)$ respectively, were $k = |W|$. And the time complexity for each new arriving task is $O(k)$, where $k = |W|$.

**Competitive Analysis.** We analyze the lower bound of the competitive ratio of Greedy-HST by considering a "bad" example and it turns out that Balance is theoretically as bad as Greedy in the lower bound of competitive ratio.

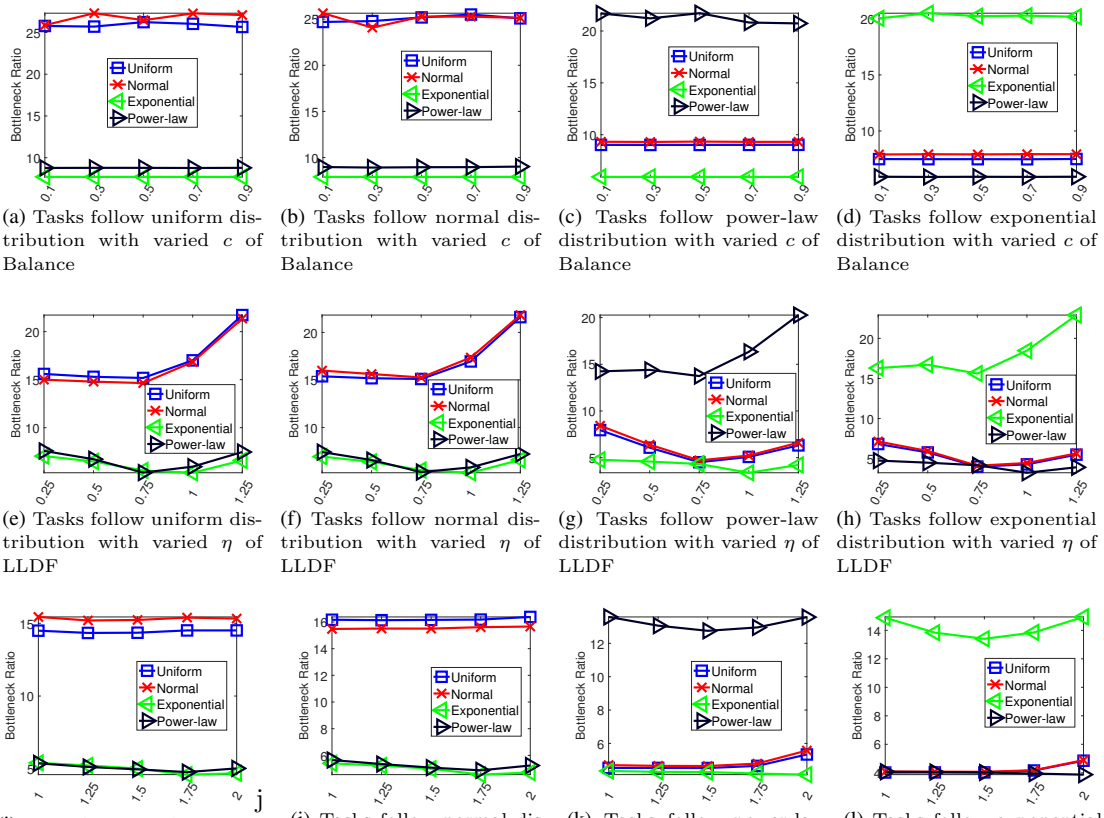
**Theorem 2.** Greedy-HST's CR in the adversarial model is at least $2^{k-1}$, where $k = |M| = \min(|T|, |W|)$.

*Proof.* Consider a case which all workers and tasks locate on a line, and $k$ workers, $\{w_0, w_1, w_2, ..., w_{k-1}\}$, locate at points $\{-\varepsilon, 2, 2^2, ..., 2^{k-1}\}$ on the line, respectively. And $\varepsilon$ is an arbitrarily small positive number. Moreover, $k$ tasks, $t_0, t_1, t_2, ..., t_{k-1}$, appear at points $\{1, 2, 2^2, ..., 2^{k-1}\}$ one by on the same line, respectively. Apparently, the optimal bottleneck matching algorithm would match $t_i$ to the corresponding $w_i$, for $i = 0, 1, ..., k-1$. And we have $Cost(OPT) = 1 + \varepsilon$.

When $t_0$ appears, Greedy-HST matches $t_0$ to $t_0$'s nearest unmatched workers, $w_1$. And then, $t_i$ is matched to $w_{i+1}$ for $i = \{1, 2, ..., k-2\}$. Finally, $t_{k-1}$ is matched to $w_0$ when it appears. The bottleneck cost of Greedy-HST is $2^{k-1} + \varepsilon$. And the competitive ratio is $\frac{2^{k-1}+\varepsilon}{1+\varepsilon} \approx 2^{k-1}$. So, we can know that Greedy-HST's CR in the adversarial model is at least $2^{k-1}$.

**Table E1** Synthetic spatial distributions' setting

| Distribution | Parameters | Scalability |
|---|---|---|
| Uniform | None | $\|T\| = \|W\| = 10,000 - 100,000$ |
| Normal | $\mu$= 50, 75, **100**, 125, 150, $\sigma$ =15, 20, **25**, 30, 35 | $\|T\| = \|W\| = 10,000 - 100,000$ |
| Power-law | $\alpha$=0.1, 0.2, **0.3**, 0.4, 0.5 | $\|T\| = \|W\| = 10,000 - 100,000$ |
| Exponential | $\lambda$=0.5, 0.75, **1**, 1.25, 1.5 | $\|T\| = \|W\| = 10,000 - 100,000$ |




(a) Tasks follow uniform distribution with varied $c$ of Balance
(b) Tasks follow normal distribution with varied $c$ of Balance
(c) Tasks follow power-law distribution with varied $c$ of Balance
(d) Tasks follow exponential distribution with varied $c$ of Balance
(e) Tasks follow uniform distribution with varied $\eta$ of LLDF
(f) Tasks follow normal distribution with varied $\eta$ of LLDF
(g) Tasks follow power-law distribution with varied $\eta$ of LLDF
(h) Tasks follow exponential distribution with varied $\eta$ of LLDF
(i) Tasks follow uniform distribution with varied $\kappa$ of LLDF
(j) Tasks follow normal distribution with varied $\kappa$ of LLDF
(k) Tasks follow power-law distribution with varied $\kappa$ of LLDF
(l) Tasks follow exponential distribution with varied $\kappa$ of LLDF

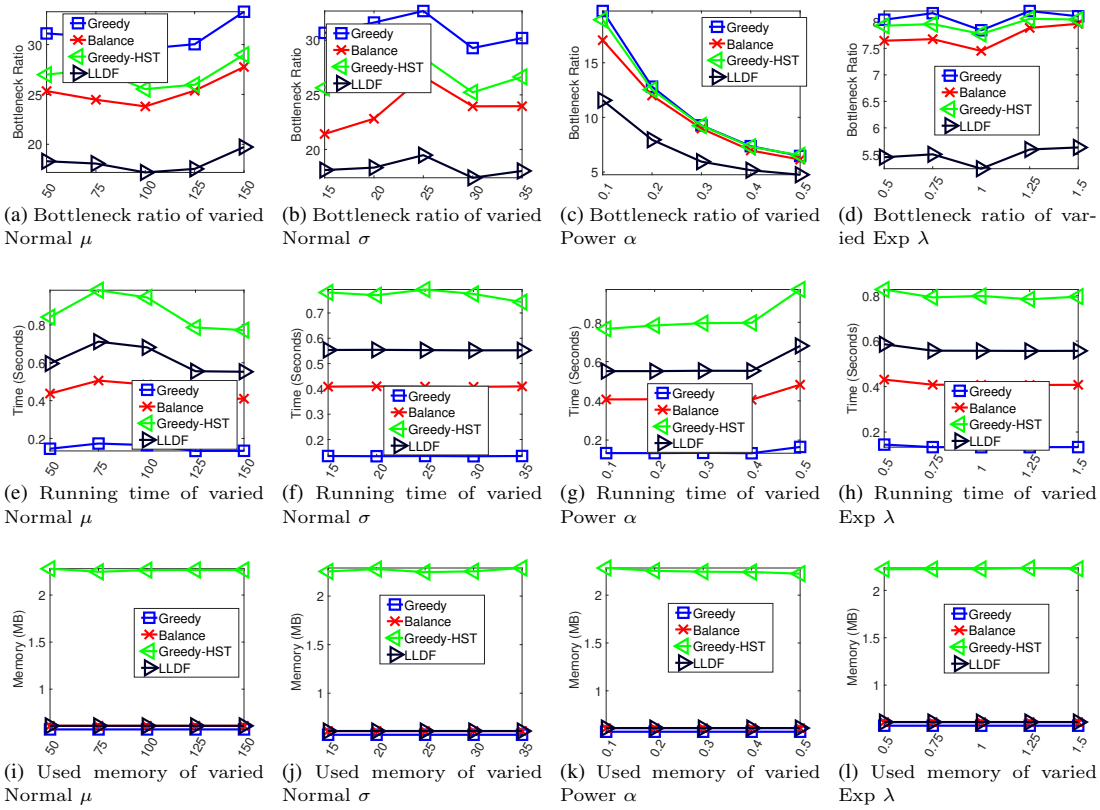
**Figure E1** Results of varied $c$ of Balance, varied $\eta$ of LLDF and varied $\kappa$ of LLDF with different spatial distributions of workers.

**Table E2** Comparison of Greedy, Permutation, Balance, Greedy-HST and LLDF with $|T| = |W| = 500$

| Distributions | Bottleneck Ratio | | | | Running Time (Seconds) | | | | Memory (MB) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | T(Uni) W(Uni) | T(Norm) W(Norm) | T(Pow) W(Pow) | T(Exp) W(Exp) | T(Uni) W(Uni) | T(Norm) W(Norm) | T(Pow) W(Pow) | T(Exp) W(Exp) | T(Uni) W(Uni) | T(Norm) W(Norm) | T(Pow) W(Pow) | T(Exp) W(Exp) |
| Greedy | 11.21 | 11.00 | 10.20 | 9.61 | 0.0019 | 0.0025 | 0.0015 | 0.0017 | 0.057 | 0.057 | 0.057 | 0.057 |
| Permutation | 10.96 | 10.70 | 10.16 | 9.38 | 32.73 | 42.07 | 27.74 | 29.51 | 5.84 | 5.84 | 5.84 | 5.84 |
| Balance | 9.54 | 8.74 | 7.90 | 7.47 | 0.0058 | 0.0074 | 0.0047 | 0.0051 | 0.057 | 0.057 | 0.057 | 0.057 |
| Greedy-HST | 9.68 | 9.63 | 8.66 | 8.41 | 0.015 | 0.019 | 0.013 | 0.013 | 0.23 | 0.23 | 0.23 | 0.23 |
| LLDF | 5.50 | 5.39 | 4.92 | 4.66 | 0.0072 | 0.0093 | 0.0057 | 0.0061 | 0.073 | 0.073 | 0.073 | 0.073 |

## Appendix E.2 Experimental Settings

**Synthetic Dataset.** The working space is a 200*200 2D grid space and the distance metric $dis(\cdot,\cdot)$ is the Euclidean distance function. And we consider four distributions, namely uniform, normal, power-law and exponential distributions, to validate LLDF's effectiveness and efficiency. The reasons why we consider these four spatial distributions are these distributions are generally used in existing online matching researches in SC [3, 13, 18, 20, 21] and [8, 14] also show the movement of customers/drivers usually follow power-law and exponential distributions. Different spatial distributions'




(a) Bottleneck ratio of varied Normal $\mu$ (b) Bottleneck ratio of varied Normal $\sigma$ (c) Bottleneck ratio of varied Power $\alpha$ (d) Bottleneck ratio of varied Exp $\lambda$

(e) Running time of varied Normal $\mu$ (f) Running time of varied Normal $\sigma$ (g) Running time of varied Power $\alpha$ (h) Running time of varied Exp $\lambda$

(i) Used memory of varied Normal $\mu$ (j) Used memory of varied Normal $\sigma$ (k) Used memory of varied Power $\alpha$ (l) Used memory of varied Exp $\lambda$

**Figure E2** Results that tasks' locations follow uniform distribution while workers' locations follow normal, power-law and exponential distributions, respectively.

specific parameters are listed in Table E1 and the bold parameters is the default parameters. Note that the default setting of scalability is $|T| = |W| = 5,000$.

**Real Dataset.** We use the online taxi-calling order dataset open-sourced by DiDi Chuxing. In the dataset, there are 7,063,532 taxi-calling orders with drop-out/pick-up time and GPS locations. And all orders are recorded in November 2016, Chengdu, China. We treat the drop-out locations as workers' locations and the pick-up locations are used as tasks' locations. Hence, there are 7,063,532 workers and 7,063,532 tasks.

In the experiments on the real dataset, we randomly sample different numbers of tasks/workers from all tasks and workers in a random order and use the GPS distance function as the distance function $dis(\cdot,\cdot)$. Note that the GPS distance is accurate into one kilometer.

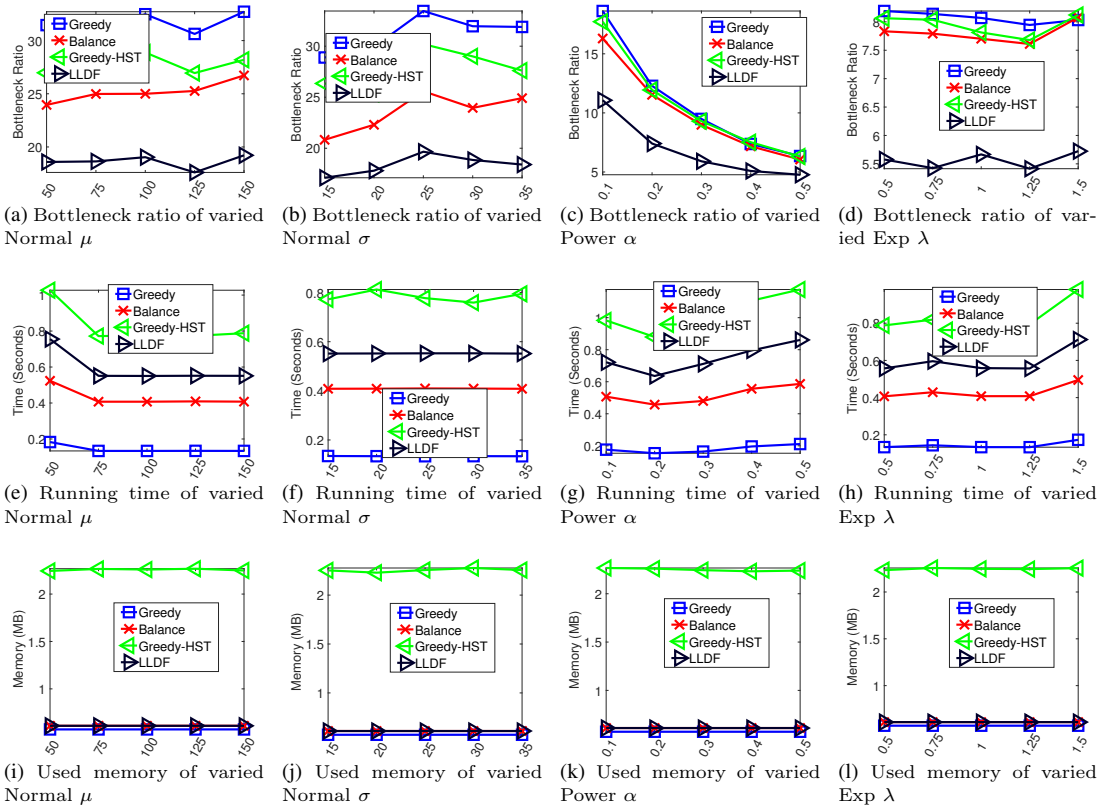
**Evaluation Metrics.**We use the following metrics to evaluate the effectiveness and efficiency of LLDF in experiments.

- **Bottleneck Ratio.** The bottleneck ratio represents the ratio of the bottleneck cost output by an online algorithm to the optimal bottleneck cost in the offline scenario and is similar with the aforementioned competitive ratio. We use the bottleneck ratio in most experiments except for the scalability experiments because the offline optimal bottleneck cost in the scalability experiments with $|T| = |W| = 10,000 - 100,000$ takes months to compute. And the bottleneck ratio has no metric unit.
- **Bottleneck Cost**. The bottleneck cost refers to the bottleneck cost output by an online algorithm and is used only for the scalability experiments with $|T| = |W| = 10,000 - 100,000$ on the synthetic and real datasets. Note that the bottleneck cost on the synthetic datasets has no metric unit and the metric unit of the bottleneck cost on the real datasets is one kilometer.
- **Running Time**: the algorithms' execution time for solving the RMBM problem.
- **Memory**: the algorithms' peak space cost for solving the RMBM problem.

**Compared Algorithms and Implementation Details.** We evaluate the performance the aforementioned online algorithms, Greedy, Permutation, Balance, Greedy-HST and LLDF on synthetic and real datasets. And in experiments, we run the algorithms repeatedly for 10 times and use the average performance as experiments' results.

Notice that Permutation is so inefficient, it takes hours for an instance with 2000 tasks/workers and the running time increase rapidly with the number of tasks/workers increasing. Hence, we have to separately compare Permutation and the other algorithms with a much smaller number of tasks/workers on the synthetic datasets ($|T| = |W| = 500$).

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




(a) Bottleneck ratio of varied Normal $\mu$ (b) Bottleneck ratio of varied Normal $\sigma$ (c) Bottleneck ratio of varied Power $\alpha$ (d) Bottleneck ratio of varied Exp $\lambda$

(e) Running time of varied Normal $\mu$ (f) Running time of varied Normal $\sigma$ (g) Running time of varied Power $\alpha$ (h) Running time of varied Exp $\lambda$

(i) Used memory of varied Normal $\mu$ (j) Used memory of varied Normal $\sigma$ (k) Used memory of varied Power $\alpha$ (l) Used memory of varied Exp $\lambda$

**Figure E3** Results that tasks' locations follow normal distribution while workers' locations follow normal, power-law and exponential distributions, respectively.

**Choice of Density Distance** *DenDis*. We use the simpler distance function as *DenDis* to accelerate LLDF.

In experiments, the Euclidean distance is used as *dis* on synthetic datasets and the GPS distance is used as *dis* on the real-world dataset. We find out that the bottleneck cost of using the simple Manhattan distance as *DenDis* is close to the bottleneck cost of using the original Euclidean distance as *DenDis* on synthetic datasets. Similarly, the bottleneck cost of using the simple Manhattan distance as *DenDis* is close to the bottleneck cost of using the original GPS distance as *DenDis* on the real dataset.

## Appendix E.3 Experiment Results

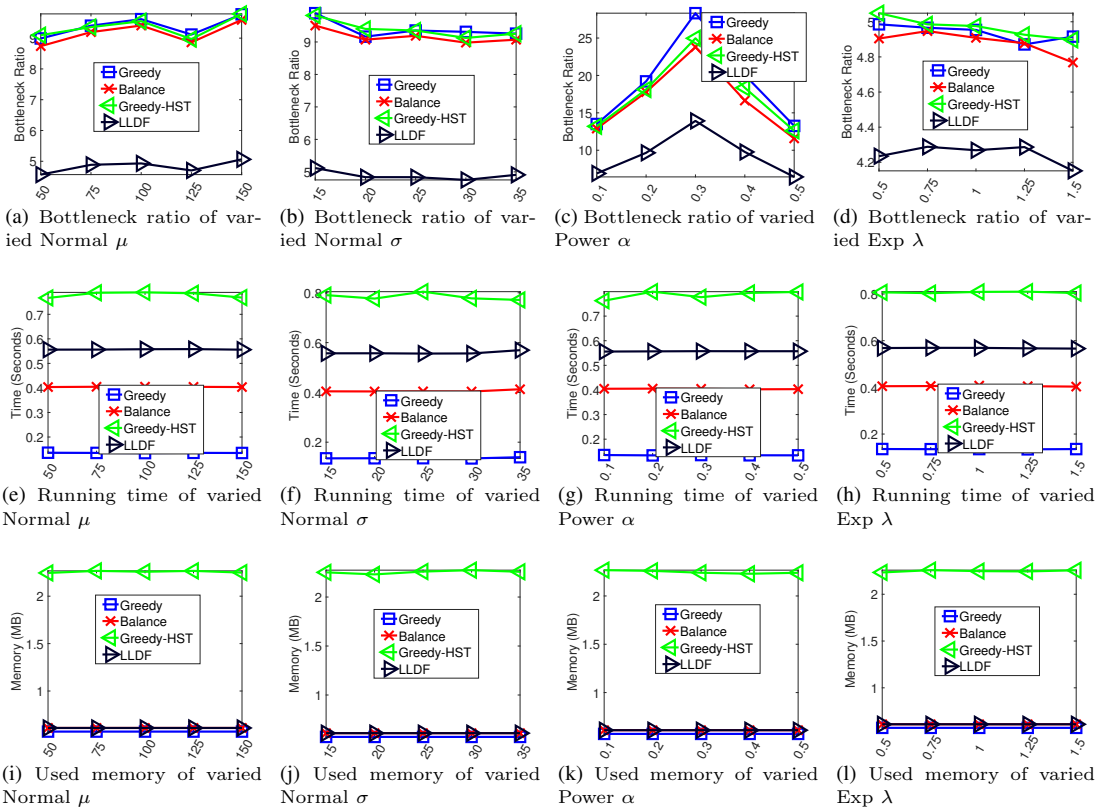
**Effect of Balance's $c$.** Fig. E1(a), Fig. E1(b), Fig. E1(d) and Fig. E1(c) show that the different settings of $c$ in Balance have little effect on the bottleneck ratio and the scalability is $|T| = |W| = 5,000$. The bottleneck ratio changes less than 10% as $c$ increasing from 0.1 to 0.9. Based on such observation, we set $c = 0.5$ by default in rest experiments.

**Effect of LLDF's $\eta$.** We set $\eta = \{0.25, 0.5, 0.75, 1, 1.25\}$ and the scalability is $|T| = |W| = 5,000$. Fig. E1(e), Fig. E1(f), Fig. E1(h) and Fig. E1(g) show that the different $\eta$ does affect the bottleneck ratio under different spatial distributions. The bottleneck ratio decreases firstly and then increases or or remains still with $\eta$ increasing from 0.25 to 1.25. And the reasonable setting of $\eta$ can reduce the bottleneck ratio. Based on this experiment, we set $\eta = 0.75$ by default in all rest experiments.

**Effect of LLDF's $\kappa$.** We set $\kappa = \{1, 1.25, 1.5, 1.75, 2\}$ and the scalability is $|T| = |W| = 5,000$. Fig. E1(i), Fig. E1(j), Fig. E1(l) and Fig. E1(k) shows the different $\kappa$ affects the bottleneck ratio less than the different $\eta$. And the bottleneck ratio remains almost the same with $\kappa$ increasing from 1 to 2 in most cases. We set $\kappa = 1.5$ by default setting in all rest experiments.

**Comparisons with Permutation.** Because the running time of Permutation is hundred thousands times of other algorithms and grows rapidly with the scalability increasing. We have to conduct an experiment especially for Permutation on a smaller dataset ($|T|$=$|W|$=500) and the results of the comparisons with Permutation are presented in Table E2. It turns out that Permutation performs poorly on the bottleneck ratio and all algorithms' bottleneck ratio in descending order is that Greedy>Permutation> Greedy-HST>Balance$\gg$LLDF. As of the running time and memory, Permutation is really inefficient. We can conclude that Permutation is both inefficient and ineffective for the RMBM problem.

**Effect of tasks' locations following uniform distribution.** In this experiment, the tasks' locations follow the uniform distribution and the workers' locations, meanwhile, follow 3 different distributions, namely normal, power-law and




(a) Bottleneck ratio of varied Normal $\mu$ (b) Bottleneck ratio of varied Normal $\sigma$ (c) Bottleneck ratio of varied Power $\alpha$ (d) Bottleneck ratio of varied Exp $\lambda$

(e) Running time of varied Normal $\mu$ (f) Running time of varied Normal $\sigma$ (g) Running time of varied Power $\alpha$ (h) Running time of varied Exp $\lambda$

(i) Used memory of varied Normal $\mu$ (j) Used memory of varied Normal $\sigma$ (k) Used memory of varied Power $\alpha$ (l) Used memory of varied Exp $\lambda$

**Figure E4** Results that tasks' locations follow power-law distribution while workers' locations follow normal, power-law and exponential distributions.

exponential distributions. And the results are shown in Fig. E2.

LLDF's bottleneck ratio is always much lower than Greedy, Balance and Greedy-HST when the parameters of workers' distribution change. The bottleneck ratios of Greedy, balance and Greedy-HST change drastically when workers follow the normal distribution when the $\mu$ or $\sigma$ changes as shown in Fig. E2(a) and Fig. E2(b). When workers' distribution is exponential or power-law, Greedy, Balance and Greedy-HST perform closely for the bottleneck ratio.

For the running time, Greedy-HST is most inefficient than other algorithms. LLDF consumes less than 0.8 second to match 5,000 tasks and workers, which is efficient but still slower than Greedy and Balance due to the initialization of LLDF.

LLDF is efficient in memory as shown in Fig. E2. Greedy, Balance and LLDF consume a similar amount of memory. Greedy-HST is inefficient because the construction of a 2-HST consumes too much space and time.

**Effect of tasks' locations following normal/power-law/exponential distribution.** The experiment results when the tasks' locations follow normal/power-law/exponential distributions while the workers' locations follow normal, power-law and exponential distributions are shown in Fig. E3/Fig. E4/Fig. E5, respectively.

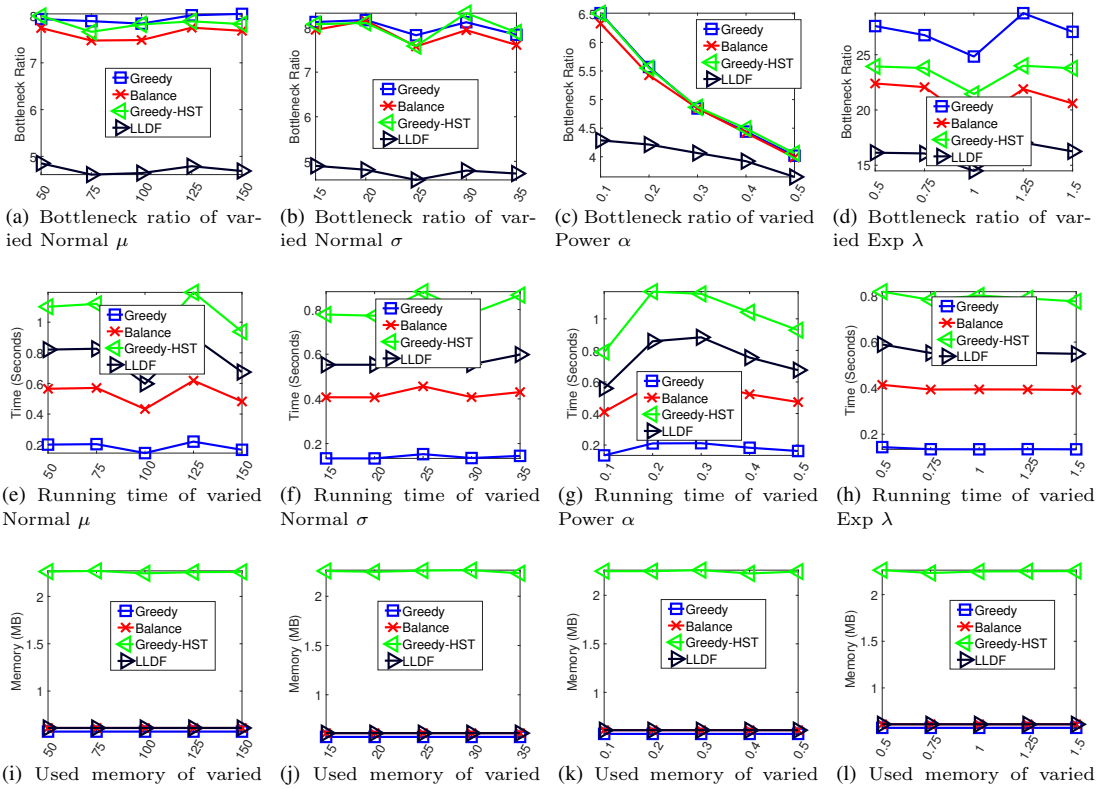
For the bottleneck ratio, LLDF performs the best. And LLDF is efficient in terms of running-time and memory. Greedy-HST consumes more running time and memory than others.

**Scalability.** The algorithms' scalability is studied and the cardinality, $|T|/|W|$, varies from 10,000 to 100,000 in this experiment. Fig. E6 shows the results of the scalability experiments when the locations of tasks/workers follow uniform, normal, power-law and exponential distributions, respectively. And Fig. E6(e), Fig. E6(j) and Fig. E6(o) presented the scalability experiments on the real dataset. As shown in these figures, LLDF always outputs the lower bottleneck cost under different spatial distributions of tasks/workers. And Greedy performs worst for the bottleneck cost in all situations.

As same as all the aforementioned experiments, LLDF also consumes less time than Greedy-HST. All 4 online algorithms' running time in descending order is Greedy-HST>LLDF>Balance>Greedy. And all 4 online algorithms' memory in descending order is Greedy-HST$\gg$LLDF$\approx$Greedy$\approx$Balance.

**Summary of aforementioneds experiments.**

- LLDF performs drastically smaller bottleneck cost than all baseline algorithms and LLDF is efficient in terms of running time and memory.
- In general, all 5 online algorithms' running time in descending order is Permutation$\gg$Greedy-HST>LLDF>Balance >Greedy.
- In general, all 5 online algorithms' consuming memory in descending order is Permutation$\gg$Greedy-HST$\gg$LLDF$\approx$ Balance$\approx$Greedy.




(a) Bottleneck ratio of varied Normal $\mu$ (b) Bottleneck ratio of varied Normal $\sigma$ (c) Bottleneck ratio of varied Power $\alpha$ (d) Bottleneck ratio of varied Exp $\lambda$

(e) Running time of varied Normal $\mu$ (f) Running time of varied Normal $\sigma$ (g) Running time of varied Power $\alpha$ (h) Running time of varied Exp $\lambda$

(i) Used memory of varied Normal $\mu$ (j) Used memory of varied Normal $\sigma$ (k) Used memory of varied Power $\alpha$ (l) Used memory of varied Exp $\lambda$

**Figure E5** Results that tasks' locations follow exponential distribution while workers' locations follow normal, power-law and exponential distributions, respectively.

## References


1 Barbara M Anthony and Christine Chung. Online bottleneck matching. *Journal of Combinatorial Optimization*, 27(1):100–114, 2014.

2 Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *SIGMOD*, volume 29, pages 93–104. ACM, 2000.

3 Zhao Chen, Peng Cheng, Yuxiang Zeng, and Lei Chen. Minimizing maximum delay of task assignment in spatial crowdsourcing. In *ICDE*, pages 1454–1465. IEEE, 2019.

4 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

5 Harold N Gabow and Robert E Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9(3):411–417, 1988.

6 Robert S Garfinkel. An improved algorithm for the bottleneck assignment problem. *Operations Research*, 19(7):1747–1751, 1971.

7 Andrew V Goldberg, Sagi Hed, Haim Kaplan, Robert E Tarjan, and Renato F Werneck. Maximum flows by incremental breadth-first search. In *ESA*, pages 457–468. Springer, 2011.

8 Zhiqiang Jiang, Wenjie Xie, Mingxia Li, Boris Podobnik, Weixing Zhou, and H Eugene Stanley. Calling patterns in human communication dynamics. *Proceedings of the National Academy of Sciences*, 110(5):1600–1605, 2013.

9 Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

10 Bala Kalyanasundaram and Kirk Pruhs. The online transportation problem. *SIAM Journal on Discrete Mathematics*, 13(3):370–383, 2000.

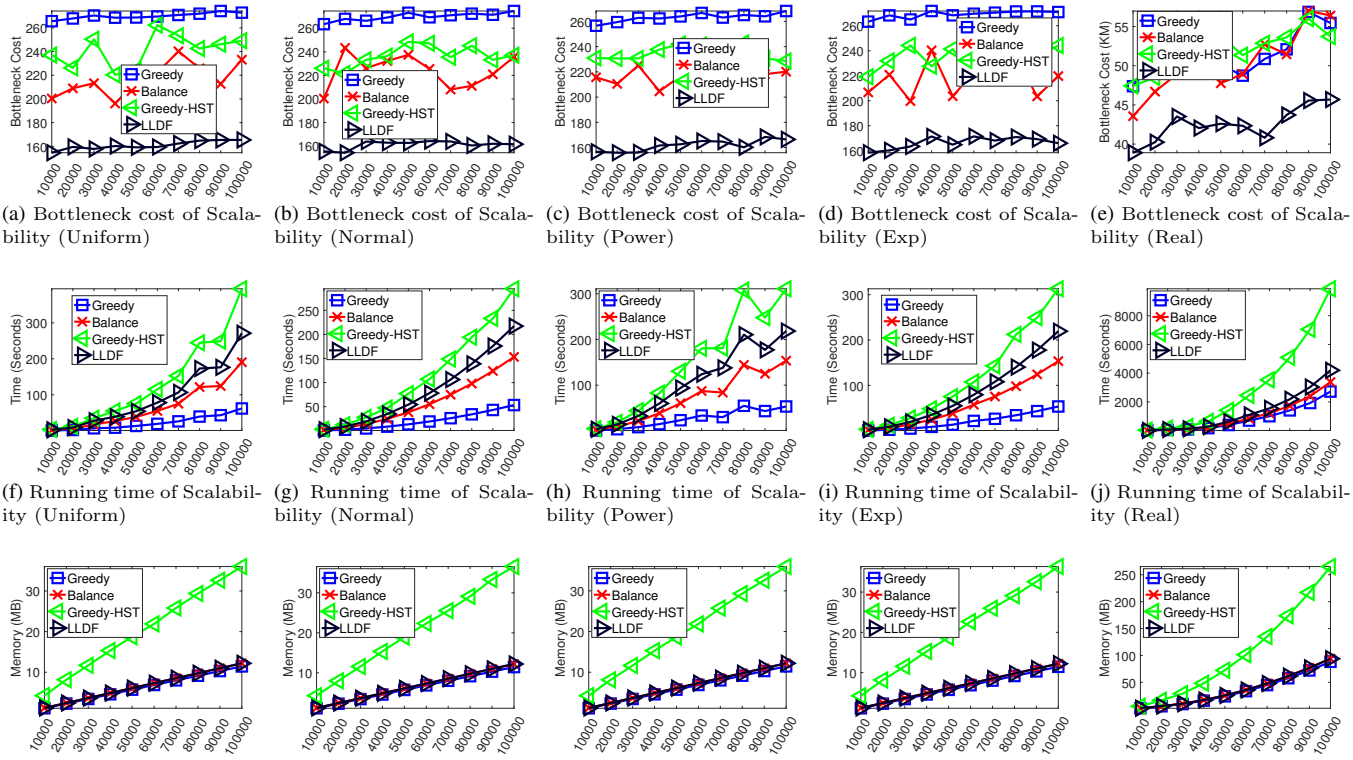
11 Leyla Kazemi, Cyrus Shahabi, and Lei Chen. Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In *GIS*, pages 314–323, 2013.

12 Samir Khuller, Stephen G Mitchell, and Vijay V Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.

13 Long Li, Jingzhi Fang, Bowen Du, and Weifeng Lv. Spatial bottleneck minimum task assignment with time-delay. In *DASFAA*, pages 387–391, 2019.

14 Xiao Liang, Jichang Zhao, Li Dong, and Ke Xu. Unraveling the origin of exponential law in intra-urban human mobility. *Scientific Reports*, 3(1):2983–2983, 2013.

15 Adam Meyerson, Akash Nanavati, and Laura Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *SODA*, pages 954–959. SIAM, 2006.




(a) Bottleneck cost of Scalability (Uniform) (b) Bottleneck cost of Scalability (Normal) (c) Bottleneck cost of Scalability (Power) (d) Bottleneck cost of Scalability (Exp) (e) Bottleneck cost of Scalability (Real)

(f) Running time of Scalability (Uniform) (g) Running time of Scalability (Normal) (h) Running time of Scalability (Power) (i) Running time of Scalability (Exp) (j) Running time of Scalability (Real)

(k) Used memory of Scalability (Uniform) (l) Used memory of Scalability (Normal) (m) Used memory of Scalability (Power) (n) Used memory of Scalability (Exp) (o) Used memory of Scalability (Real)

**Figure E6** Results of scalability when the locations of tasks/workers follow uniform, normal, power-law and exponential distributions and the results of scalability on the real dataset.


16 Ninh Pham and Rasmus Pagh. A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data. In *SIGKDD*, pages 877–885. ACM, 2012.

17 Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*, volume 29, pages 427–438. ACM, 2000.

18 Tianshu Song, Yongxin Tong, Libin Wang, Jieying She, Bin Yao, Lei Chen, and Ke Xu. Trichromatic online matching in real-time spatial crowdsourcing. In *ICDE*, pages 1009–1020, 2017.

19 Hien To, Cyrus Shahabi, and Leyla Kazemi. A server-assigned spatial crowdsourcing framework. *ACM Transactions on Spatial Algorithms and Systems*, 1(1):2, 2015.

20 Yongxin Tong, Jieying She, Bolin Ding, Lei Chen, Tianyu Wo, and Ke Xu. Online minimum matching in real-time spatial data: experiments and analysis. *PVLDB*, 9(12):1053–1064, 2016.

21 Yongxin Tong, Jieying She, Bolin Ding, Libin Wang, and Lei Chen. Online mobile micro-task allocation in spatial crowdsourcing. In *ICDE*, pages 49–60. IEEE, 2016.

22 Yongxin Tong, Libin Wang, Zimu Zhou, Bolin Ding, Lei Chen, Jieping Ye, and Ke Xu. Flexible online task assignment in real-time spatial data. *PVLDB*, 10(11):1334–1345, 2017.