Privacy-Preserving Batch-based Task Assignment in Spatial Crowdsourcing with Untrusted Server

Maocheng Li  
csmichael@cse.ust.hk  
HKUST  
Hong Kong, China

Jiachuan Wang  
jwangey@cse.ust.hk  
HKUST  
Hong Kong, China

Libin Zheng  
zhenglb6@mail.sysu.edu.cn  
Sun Yat-sen University  
Guangzhou, China

Han Wu  
han.wu@stu.ecnu.edu.cn  
East China Normal University  
Shanghai, China

Peng Cheng  
pcheng@sei.ecnu.edu.cn  
East China Normal University  
Shanghai, China

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

Xuemin Lin  
xue@cse.unsw.edu.au  
The University of New South Wales  
Sydney, Australia

ABSTRACT

In this paper, we study the privacy-preserving task assignment problem in spatial crowdsourcing, where the locations of both workers and tasks, prior to their release to the server, are perturbed with Geo-Indistinguishability (a differential privacy notion for location-based systems). Different from the previously studied online setting, where each task is assigned immediately upon arrival, we target the batch-based setting, where the server maximizes the number of successfully assigned tasks after a batch of tasks arrive. To achieve this goal, we propose the $k$-Switch solution, which first divides the workers into small groups based on the perturbed distance between workers/tasks, and then utilizes Homomorphic Encryption (HE) based secure computation to enhance the task assignment. Furthermore, we expedite HE-based computation by limiting the size of the small groups under $k$. Extensive experiments demonstrate that, in terms of the number of successfully assigned tasks, the $k$-Switch solution improves batch-based baselines by 5.9$x$ and the existing online solution by 1.74$x$, with no privacy leak.

CCS CONCEPTS
• Security and privacy → Domain-specific security and privacy architectures; • Information systems → Location based services; Crowdsourcing.

1 INTRODUCTION

The mass adoption of GPS-equipped smart phones enables individuals to collaborate, participate, consume and produce valuable information about the environment and themselves. Spatial crowdsourcing (SC) systems (e.g., Foursquare [1], Gigwalk [2], MediaQ [13], and gMission [6]) have emerged to support such collaborations by assigning tasks to proper workers. Task assignment is a core issue in SC systems, asking the workers to move physically to specified locations to execute the tasks [12, 22]. To enable the SC server to properly assign tasks, in general, workers need to upload their locations. However, users’ location is highly sensitive, because it can indicate users’ whereabouts, and even disclose their private attributes. For example, visiting an urgent care center reveals certain medical conditions [9]. The server, which receives the locations, is untrusted and can be vulnerable to attacks. Thus, in previous studies [18, 21], privacy-preserving task assignment is proposed to enable users (workers/task requesters) to perturb their locations with Geo-Indistinguishability (Geo-I) [5] and upload only the perturbed locations. Geo-I is a widely adopted differential privacy notion [8] for location-based systems, which defends users’ locations against strong adversaries with any prior knowledge [9].

In this paper, we study the task assignment problem with users’ perturbed locations under the batch-based setting. That is, the SC server assigns the tasks batch-by-batch to the workers, with locations of both workers and tasks perturbed by Geo-I. Although Geo-I is an existing technique to protect location privacy of users, directly applying Geo-I to task assignment problem could lead to poor performance as measured by the number of successfully assigned tasks, because only the perturbed locations are available as the problem inputs. To the best of our knowledge, existing works on spatial crowdsourcing with Geo-I [18, 21] target the online setting where tasks come on the fly and are processed one by one. Batch-based task assignment with Geo-I-perturbed locations has not yet been studied. Directly applying the online assignment methods to the batch-based setting can result in poor solutions, because online solution assigns tasks one by one, and previous assignments can not be changed when new tasks arrive. An example of the online solution making sub-optimal assignments is shown in Figure 1.
Reachable distance for a worker

An assignment (matching) between workers and tasks

A baseline matching obtained by oblivious baseline methods

Privacy parameters

Description

Euclidean distance function

A worker and a task after perturbation

\( p \) True locations for a worker

\( w' \) Perturbed locations to obtain an initial assignment, then uses

\( w \) Optimal solution

\( t \) Reachable distance for a worker

\( f \) True locations for a worker and a task

\( f' \) Perturbed locations for a worker and a task

\( R_w \) Reachable distance for a worker

\( f, f' \) Worker and a task after perturbation

\( f_w, f_t \) True locations for a worker and a task

Our Privacy-preserving Batch-based Task Assignment

Figure 1: Online solution makes sub-optimal assignment vs. Optimal assignment.

As Fig. 1(a) shows, in the online setting, task \( t_1 \) arrives first and
would be assigned to worker \( w_1 \) since they are close to each other.
Note that such a decision is irrevocable in the online setting. Subse-
quently, \( w_1 \) will no longer be available no matter what future tasks
arrive. When task \( t_2 \) arrives later, because worker \( w_1 \) is occupied,
\( t_2 \) cannot be assigned. \( w_1 \) is the only worker who can perform \( t_2 \),
as \( t_2 \) is outside the reachability range \( R_w \) of worker \( w_2 \). In contrast,
when \( t_1 \) and \( t_2 \) come in a batch, they can be properly assigned to
\( w_2 \) and \( w_1 \), respectively.

No existing research has specifically targeted the batch-based
privacy-preserving task assignment, however, as a matter of fact,
batch-based setting has been widely adopted in the spatial crowd-
sourcing industry. For example, DiDi [3], the leading car-hailing
platform in China, accumulates orders (tasks) per time window,
and jointly dispatches them to drivers (workers) [26, 29]. Figure 2
gives an example of our studied problem in this paper. Fig. 2(a)
shows the true locations of workers \( w_1, w_2 \) and tasks \( t_1, t_2 \). \( w_1 \) and
\( w_2 \) have a reachability range \( R_w \) and \( R_w \) respectively, denoting
the maximum distance that they are willing to travel. In Fig. 2(b),
the locations are perturbed using Geo-I. Our problem is to assign
tasks to suitable workers to maximize the number of assigned tasks
subjecting to workers’ reachability constraint.

The obfuscated locations bring technical challenges to the task
assignment problem. As shown in Fig. 2(b), if we directly use the
perturbed locations as the true locations to perform the task as-
signment, then task \( t_2 \) and \( t_1 \) are assigned to worker \( w_2 \) and \( w_1 \)
respectively (denoted by the red arrows). However, according to
the true locations, \( t_2 \) lies outside the range of \( w_2 \), and \( t_1 \) lies outside
the range of \( w_1 \). The aforementioned assignment finally has no task
successfully executed. In fact, the optimal assignment should be
assigning \( t_2 \) to \( w_2 \) and \( t_1 \) to \( w_1 \) (denoted by the green arrows in
Fig. 2(a)), which would finally execute both tasks.

To address the aforementioned challenge, we propose an innova-
tive solution \( k \)-Switch. It first employs probabilistic inference over
the perturbed locations to obtain an initial assignment, then uses
an encryption-based secure computation method to improve the
assignment quality significantly without compromising privacy. To
address the efficiency issue of existing Homomorphic Encryption
techniques, it divides workers into small groups of size \( k \), and apply
HE within the small groups only. Using the example in Fig. 2, \( k \)-
Switch uses perturbed locations to obtain a preliminary assignment
first, and then allows worker \( w_1 \) and \( w_2 \) to use secure communica-
tion to obtain the true distance from the tasks and switch tasks in
order to obtain a better assignment.

To summarize, we make the following contributions in this paper.

- We propose \( k \)-Switch, an innovative solution to address the chal-
  lenging Privacy-preserving Batch-based Task Assignment (PBTA)
  problem. To the best of our knowledge, this is the first work
targeting the batch-based setting of the privacy-preserving task
assignment problem. The PBTA problem is defined in Section 3
together with two proposed oblivious baselines.
- On top of the oblivious baselines, we propose the \( k \)-Switch method,
  which combines probabilistic analysis and encryption-based se-
cure computation techniques to achieve significant assignment
  quality improvements without compromising privacy. It miti-
ge the efficiency issue of previous encryption-based secure
  computation techniques by doing the computation within the
  small groups. We introduce \( k \)-Switch in Section 4.
- We conduct extensive experiments to validate the efficiency and
  effectiveness of the proposed algorithms. In terms of the number
of successfully assigned tasks, the \( k \)-switch method assigns up to
5.9× more tasks than other proposed baselines, and assigns up to
1.74× more tasks than the existing online solution. The results
are shown in Section 5.

In addition, we cover the preliminaries and necessary back-
ground in Sec. 2, compare with other related works in Sec. 6 and
conclude the paper in Sec. 7. The notations used in this paper are
summarized in Table 1.

### Table 1: Notation.

| Symbol | Description |
|--------|-------------|
| \( w, t, W, T \) | A worker, a task, a set of workers, a set of tasks |
| \( w' \) | A worker and a task after perturbation |
| \( f_w, f_t \) | True locations for a worker \( w \) and a task \( t \) |
| \( f'_w, f'_t \) | Perturbed locations for a worker \( w \) and a task \( t \) |
| \( R_w \) | Reachable distance for a worker \( w \) |
| \( f, f' \) | Worker and a task after perturbation |
| \( M \) | An assignment (matching) between workers and tasks |
| \( M_o \) | A baseline matching obtained by oblivious baseline methods |
| \( p, p' \) | \( p \) is a matched worker/task pair \( p = (w, t) \). \( P = W \times T \) is the set of all possible worker-task pairs. |
| \( t = e \) | Privacy parameters |

### 2 BACKGROUND

#### 2.1 Task assignments in spatial crowdsourcing

We introduce the task assignment problem, especially the batch-
based setting, which is the primary focus of this paper.

**Definition 2.1.** (Task assignment problem in spatial crowdsourc-
ing [22, 27]) Given a set of workers \( W \) and a set of tasks \( T \), the
task assignment problem returns an assignment (matching) \( M \) of
tasks to workers \( M = \{(w, t) | w \in W, t \in T \} \) such that for some
given objective function \( \Psi(M) \), \( \Psi(M) \) is optimized (maximized or
minimized):

\[
\Psi(M) = \sum_{(w, t) \in M} \psi(w, t),
\]
Following the setting in [21], we have the following spatial constraint and objective function. Each worker is willing to travel at most $R_w \in \mathbb{R}$, i.e., for each pair $(w, t) \in M$, $d(l_w, l_t) \leq R_w$. Each successfully assigned task carries a unit utility. The task assignment problem maximizes $\Psi(M) = |M|$, the size of the matching.

In this paper, we focus on the batch-based setting (also referred as offline or static setting): the locations of all workers and tasks are known at the beginning. Previous works [18, 21] mainly focus on the privacy preservation for the online setting: tasks arrive one by one, and each task needs to be assigned immediately upon arrival and cannot be re-assigned to other workers no matter what future tasks arrive.

2.2 Geo-indistinguishability

Geo-indistinguishability (Geo-I) [5] extends the traditional and well-adopted privacy notion – differential privacy [8] to location-based systems.

Definition 2.2. (Geo-I) For all true locations $x, x'$, a privacy parameter $\epsilon$, a mechanism $M$ satisfies $\epsilon$-Geo-I if:

$$d_{\epsilon}(M(x), M(x')) \leq ed(x, x'),$$

where $d(x, x')$ is the Euclidean distance between $x$ and $x'$ while $d_{\epsilon}(M(x), M(x'))$ is the multiplicative distance between two distributions $M(x)$ and $M(x')$. $M(x)$ and $M(x')$ are the distributions of perturbed locations based on the original location $x$ and $x'$ respectively.

One particular mechanism satisfying Geo-I is drawing random noise from the planar Laplace distribution [5]. Given the privacy parameter $\epsilon \in \mathbb{R}^+$, the actual location $x_0 \in \mathbb{R}^2$, the probability density function of a noisy location $x \in \mathbb{R}^2$ is:

$$D_{\epsilon}(x_0)(x) = \frac{\epsilon^2}{2\pi} e^{-\epsilon d(x_0, x)},$$

where $\frac{\epsilon^2}{2\pi}$ is the normalization factor.

3 PROBLEM DEFINITION

We formally define the Privacy-preserving Batch-based Task Assignment (PBTA) problem in this section and introduce our proposed oblivious baselines.

3.1 PBTA problem

Definition 3.1. (Privacy-preserving Batch-based Task Assignment (PBTA) problem) Given a set of workers $W$ and a set of tasks $T$, the perturbed location $l_w'$ for each task $w$, the reachable distance $R_w$ for each worker $w$, the Euclidean distance function $d(\cdot, \cdot)$, the PBTA problem is to return an assignment (matching) $M$ of tasks to workers $M = \{(w, t) | w \in W, t \in T, d(l_w, l_t) \leq R_w\}$ such that the following objective function is maximized:

$$\Psi(M) = |M|.$$

Note that in the definition above, only the perturbed locations $l_w', l_t'$ are available in the input. On the other hand, for the objective we are maximizing, each pair in the matching $M$ needs to satisfy the spatial constraint w.r.t. the true locations $l_w, l_t$. We have shown a simple example in Sec. 1 (Figure 2) to illustrate the problem.

In our setting, The perturbed locations of both workers $l_w'$ and tasks $l_t'$ are obtained by applying Geo-I with the privacy level $\epsilon = er$. The privacy level is the same for all workers and tasks. Also, one worker $w$ takes at most one task, and each task $t$ only needs to be matched to one worker. The setting follows [21].

3.2 Privacy model

3.2.1 System model. We follow the system model assumptions of [21]. We have three parties in our system: the server, the workers, and the task requesters (short as tasks hereafter).

Because the server is untrusted, when workers and tasks submit their locations, they only send the perturbed locations. The server is untrusted, as centralized servers are usually vulnerable to attacks, suffering massive data leak. For example, Facebook security breach exposes 50 million users’ data [11].

The task assignment is done at the server. After workers and tasks submit perturbed locations to the server, the server runs its task assignment algorithm, obtains an assignment $M$, and notifies workers and tasks about their assignments.

3.2.2 Adversary model. Similar to the setting in [21], we adopt a semi-honest model, which assumes that all participating parties (the workers, the task requesters, and the server) are curious but not malicious.

They are curious about the private and sensitive information about other parties. So, we try to prevent sensitive information of any party from being shared with other parties. On the other hand, they are not malicious and follow system protocols. They do not collude with each other to gain extra information.

3.3 Baselines

We propose a simple baseline solution Oblivious-M, which directly uses the perturbed (observed) locations to obtain a matching between workers and tasks. Then, we incorporate probabilistic analysis of reachability between workers and tasks, and propose another solution Oblivious-RR, which is based on randomized rounding.

The term ‘Oblivious’ indicates that both of these methods only access the perturbed locations, and true locations have never been accessed and disclosed in any way when running these methods.

3.3.1 Oblivious-M. The Oblivious-M method is the simplest baseline solution we propose. It builds the reachability graph from the perturbed locations, and runs the Max-Flow algorithm (e.g., Ford-Fulkerson algorithm [7]) to obtain a maximized cardinality matching.

We use a running example in Figure 3 (modified from Fig. 3 of [21]) to demonstrate the basic steps of Oblivious-M. In the PBTA problem, we are given the observed (perturbed) locations of workers and tasks (Fig. 3(a)). In Oblivious-M method, directly using the observed locations, if a task $t$ is within the range $R_w$ of worker $w$, we create an edge between $w$ and $t$ indicating they are reachable. In this way, we build the reachability graph as shown in Fig. 3(b).

The reachability graph is a bipartite graph with all workers on one side and tasks on the other side, and all edges (indicating reachability) in between. Next, we simply add superficial source/sink nodes, edges with capacity 1 between the source node and every
worker, and edges with capacity 1 between the sink node and every task to build the flow network (see Fig. 4). It is shown that the maximum cardinality matching from the reachability graph corresponds to the Max-Flow on the constructed flow network. So, we run standard Max-Flow algorithms (such as Ford-Fulkerson [7]) on the constructed flow network, and return all saturated edges \((w, t)\) (with flow value 1 on the edges) as the output \(M\). In our running example, the maximum flow obtained is shown in Fig. 4. The returned matching is \(M = \{(w_1, t_2), (w_2, t_1), (w_3, t_3)\}\).

When we measure the size of the matching \(M\), we need to use the true locations to check whether a task is indeed within the reach of the assigned worker. The true locations and the reachability graphs built from the true locations are shown in Figure 5. In our returned matching \(M\) based only on the perturbed locations, \(t_2\) and \(t_1\) are indeed within the range of \(w_1\) and \(w_2\) respectively. However, \(t_3\) lies outside the range of \(w_3\). Thus, the size of the matching we find is 2 (instead of 3).

For the time complexity, the fattest-path first implementation of Ford-Fulkerson is \(O(|E| \cdot opt)\), where \(E\) denotes edges for the flow network, and \(opt\) is the optimal value of the flow. We let \(n = \max(|W|, |T|)\), the larger value of the sizes of the workers and the tasks. In our constructed flow network, assuming all tasks are connected with all workers in the worst case, \(|E| = O(n^2)\). As for \(opt\), the optimal flow value is bounded by \(n\). Overall, Oblivious-M has complexity of \(O(|E| \cdot opt) \rightarrow O(n^2 \cdot n) \rightarrow O(n^3)\).

### 3.3.2 Oblivious-RR

When we construct the reachability graph in the simplest baseline solution Oblivious-M, an edge between a worker \(w\) and a task \(t\) is either 1 (if perturbed locations show that \(t\) is reachable from \(w\)) or 0 (when \(t\) is not reachable from \(w\)). We make improvements in this step by adopting probabilistic analysis. Intuitively, based on perturbed locations, some worker-task pairs should have higher likelihood that they are indeed reachable, and on the other hand, some other worker-task pairs should have lower likelihood of being reachable.

We continue with the running example in Figure 3 to further explain the intuition of the probabilistic analysis. If we focus on \(w_1\), and compare two tasks \(t_1\) and \(t_2\), then based on the perturbed locations, \(t_2\) looks much closer to \(w_1\) compared to \(t_1\) (the distance between \(w_1\) to \(t_1\) is almost \(1/2\) of \(w_1\) to \(t_2\) as \(t_2\) is almost at the periphery of the reachable circle). Oblivious-RR incorporates the probabilistic analysis and gives each edge in the reachability graph a fractional weight ranging from 0 to 1, instead of a binary 0/1 value. Based the fractional flow network, we design randomized techniques to obtain a matching. Due to space limit, we leave the details of Oblivious-RR in the appendix (Sec. A.1). Here, we show a particular matching obtained after the randomized rounding in Fig. 6(b), which assigns the same amount of tasks as the optimal matching.

The time complexity of Oblivious-RR is the same with Oblivious-M, as it only adds a post randomized rounding which takes \(O(n^2)\) time, with \(n = \max(|W|, |T|)\). Overall, Oblivious-RR runs in \(O(n^3)\) time.

### 4 K-SWITCH

We present the main contribution of this paper – our proposed \(k\)-Switch in this section. We introduce the basic idea of the entire solution first, and then present the technical details in each step of the method.

#### 4.1 Overview

\(k\)-Switch method is a novel task swapping method, which uses coordination between workers to achieve utility gain. It trades off modest system overhead with secure computation between workers to increase the number of successfully assigned tasks, without privacy leak.

Figure 7 shows the basic idea of the method. \(k\)-Switch starts with a baseline matching obtained by our proposed Oblivious baselines, and then makes an improvement on the matching with the following steps:

- **Step 1)** Grouping (Sec. 4.3): carefully groups workers into groups of size \(k\), where \(k\) is a small number selected by the server.
- **Step 2)** \(k\)-HE (Sec. 4.4): for each \(k\)-group, the workers and the task inside the group uses secure computation to calculate the true distance based on encrypted true locations. Then, workers inside the \(k\)-group swap tasks if such swapping achieves utility gain, i.e., increase the number of successfully assigned tasks.
4.2 Baseline matching

$k$-Switch starts with a baseline matching $M_0$ obtained by either Oblivious-M or Oblivious-RR proposed in Section 3. According to the experiment results, we find that Oblivious-M outperforms Oblivious-RR in terms of the number of successfully assigned tasks, so we use it in $k$-Switch. In the future, if other better Oblivious matching methods are proposed, they could also be incorporated to find the baseline matching $M_0$.

For clarification of presentation, we formally define the baseline matching $M_0$.

**Definition 4.1.** (Baseline matching) For a PBTA problem as defined in Def. 3.1, we use Oblivious-M to find a baseline matching $M_0$.

$$M_0 = \{(w,t)|w \in W, t \in T\}$$

Again, to illustrate our idea clearly, we continue with the running example in Figure 4. Using Oblivious-M, we obtain a baseline matching $M_0 = \{(w_1,t_2), (w_2,t_1), (w_3,t_3)\}$. At this stage, only perturbed locations shown in Figure 3 is observable to the server.

As we illustrate previously, this baseline matching could be sub-optimal when we use true locations to verify the reachability constraints of the workers. In the next step Grouping, we divide workers into small groups of size $k$ for further optimization.

4.3 Grouping

4.3.1 Definitions and $k$-Grouping problem. The purpose of grouping is to group nearby workers and tasks into small groups, such that there are good chances that swapping tasks inside the small group could lead to improvement of the number of successfully assigned tasks (the utility of the grouping). The small groups are referred as $k$-groups. Next, we formally define the $k$-group and the $k$-Grouping problem.

For clarification of presentation, we use $P = W \times T$ to denote the set of all possible matched worker/task pairs. For example, in the baseline matching $M_0$ in our running example, we have three matched pairs, $p_1 = (w_1,t_2) \in P$, $p_2 = (w_2,t_1) \in P$, $p_3 = (w_3,t_3) \in P$. The baseline matching could also be denoted as $M_0 = \{p = (w,t)|w \in W, t \in T\}$.

**Definition 4.2.** ($k$-group) A $k$-group is a set of $k$ matched worker-task pairs in the baseline matching. A $k$-group $g$ contains $k$ different workers and their matched tasks:

$$g = \{p = (w,t)|p \in M_0\} \text{ s.t. } |g| = k.$$ 

Back to the running example, if $k = 2$, then a $k$-group becomes a 2-group which contains 2 matched pairs. For example, $g_1 = (p_1, p_2)$ is a possible 2-group, $g_2 = (p_1, p_3)$ is another possible 2-group. For 2-group $g_1$, it contains worker from the matched pair $p_1$, which is $w_1$. It also contains the worker from the matched pair $p_2$, which is $w_2$. The 2-group $g_1$ also contains the tasks that are matched with the workers, which are task $t_2$ in pair $p_1$ and $t_1$ in pair $p_2$.

The purpose of grouping is to group nearby workers and tasks together such that there are good opportunities that swapping tasks between them leads to better assignment. Some $k$-groups are better than the others. To evaluate the quality of each $k$-group, we propose a simple yet effective measurements, verified by experiments, called obfuscation-score (short as OScore). We first define the OScore on a 2-group and then extend it to $k$-group.

**Definition 4.3.** (Obfuscation-score) Given a 2-group $g = \{p_1, p_2\}$, we define the obfuscation-score (or OScore) of the 2-group as:

$$OScore(g) = d(t'_1, w_2, t'_2, w) + d(t'_1, t'_2, t_1, t).$$

Here $p_1$ and $p_2$ are the two matched pairs inside the 2-group $g$. Notation $p, w$ and $t$ respectively denote the worker and the task from the matched pair.

The OScore definition is straightforward, it measures two distance: i) the distance between the perturbed locations of the two workers in the 2-group; and ii) the distance between the perturbed locations of the two tasks that are assigned to the two workers in the baseline matching $M_0$. Then OScore is the sum of the two distance.

Intuitively, if both workers and their assigned tasks appear to be close to each other, they are spatially clustered, and should be grouped together in a $k$-group. Other measures more sophisticated than OScore could be defined and used, however we adopt this OScore as it is very efficient to compute and yet effective, as to be demonstrated in our experiments. Next, we extend the OScore measurement from 2-group to $k$-group.

**Definition 4.4.** (OScore of a $k$-group) For a $k$-group $g = \{p_1, \ldots, p_k\}$, we define its OScore as:

$$OScore(g) = \sum_{1 \leq i < j \leq k} OScore((p_i, p_j)).$$

For a $k$-group $g$, its OScore is the sum of the OScore of $g$’s subsets of size 2. We abuse notations to use the same OScore to refer to different definitions for 2-group and $k$-group when $k \geq 3$.

Before we introduce the $k$-grouping problem, we define the term $k$-division as a collection of $k$-groups that we select out of the baseline matching $M_0$.

**Definition 4.5.** ($k$-division) Given a baseline matching $M_0 = \{p|p \in P\}$, a $k$-division $D = \{g_1, \ldots, g_d\}$ is a set of non-overlapping $k$-groups of $M_0$, where the size of the $k$-division is $d = \lceil |M_0|/k \rceil$. We define the score of a $k$-division as:

$$score(D) = \sum_{1 \leq i \leq d} OScore(g_i).$$
Remark: i) for $1 \leq i \leq d$, $|g_i| = k$ except for at most one group, when $|M_0|$ is not a multiple of $k$; ii) $k$-groups are non-overlapping (disjoint) when workers from any $k$-group is different from any other $k$-groups.

The $k$-division is a collection of $k$-groups from the baseline matchings. It divides workers into small $k$-groups, each with exact size $k$, except for at most one group. The exception happens when the total number of workers is not a multiple of $k$, and the exceptional group has a size in the range of $[0, k-1]$. The score of the $k$-division is defined as the sum of OScore of all its $k$-groups.

Next we are ready to formally define the $k$-Grouping problem. It divides the workers in the baseline matching $M_0$ into small groups to create a $k$-division as defined in Def. 4.5, and minimizes the score of the $k$-division.

**Definition 4.6.** ($k$-Grouping problem (KG)) Given a matching $M_0 = \{p | p \in P\}$, the $k$-grouping problem returns a $k$-division $D^*$ with minimal score. Formally, for any other $k$-division $D$:

\[
\text{score}(D^*) \leq \text{score}(D).
\]

Next, we provide theoretical analysis on the hardness of $k$-Grouping problem (KG). We show for $k = 2$, KG is polynomial-time solvable, while for $k \geq 3$, it is hard to approximate.

### 4.3.2 Algorithms for $k = 2$.

**Theorem 4.7.** When $k = 2$, KG is in P class, solvable in polynomial time.

**Proof.** The overall idea of the proof is that, when $k = 2$, the KG is equiv. to finding the maximum weight matching on a general graph. We construct a graph $M_G$ from the baseline matching $M_0 = \{p | p \in P\}$ as follows. For each matched pair $p \in P$, we create a vertex $p$ and add it to graph $M_G$. We create an edge between any two vertices $p$. It is a complete graph. We define the weight of the edges as follows. For two vertices $p_1 \in M_0$ and $p_2 \in M_0$, we set the edge weight $w(p_1, p_2) = \text{OScore}(\{p_1, p_2\})$, where OScore$(\{p_1, p_2\})$ is defined in Def. 4.3.

We could then show that the maximum weight matching on $M_G$ corresponds to the optimal solution to the KG when $k = 2$. We defer the details of the proof to the appendix (Sec. A.2).

**Theorem 4.8.** When $k = 2$, KG is solvable with time complexity $O(|M_0|^2 \cdot 37)$.

**Proof.** For the maximum weight matching problem on a general graph, it is shown that it is among the hardest problem that could be solved in polynomial time, with $O(|V|^3)$ time complexity [10], where $V$ denotes the vertex set of the graph. In our setting, $|V| = |M_0|$, the size of the baseline matching.

In our setting, as the constructed graph $M_G$ is a complete graph, and convertible to bipartite graph via a simple graph transformation method (please refer to the appendix, Sec. A.2). There exists matrix multiplication algorithms on the transformed bipartite graph to obtain the maximum weight matching with time complexity $O(|V|^2 \cdot 37)$. Because the graph transformation doubles the number of vertices of the original graph $M_G$, we have $|V'| = 2 \cdot |V| = 2|M_0|$. Overall, KG problem is solvable in $O(|V'|^2 \cdot 37) \rightarrow O(2^2 \cdot 37 \cdot |V|^2 \cdot 37) \rightarrow O(|M_0|^2 \cdot 37)$. If we let $n = \max(|W|, |T|)$, because $|M_0|$ is bounded by $n$, then KG is solvable in $O(n^2 \cdot 37)$.

### Algorithm 1: Greedy-Grouping

**Input:** A baseline matching $M_0$, Perturbed locations $l_{w'}$ and $l_t$ for each $w \in W$ and $t \in T$.

**Output:** A $k$-division $D$.

1. Initialize a heap $h$, an empty set $D$
2. foreach $p_1 \in M_0$ do
   3. foreach $p_2 \in M_0$ do
      4. if $p_1.w \neq p_2.w$ then
         5. oscore := $d((p_1.w, p_1.t), (p_2.w, p_2.t))$
         6. $h$.insert($\{p_1, p_2\}$, oscore)
   7. $d := \lceil|M_0|/k\rceil$ // Calculate how many $k$-groups
5. foreach $i \in [0, d-1]$ do
   6. $g := \{\}$
   7. while $g$.size() < $k - 1$ do
      8. $p_1, p_2 := h$.pop()
      9. $g$.insert($\{p_1, p_2\}$)
   10. Mark $p_1.w$ or $p_2.w$ as used
11. $D$.insert($g$)
12. return $D$

### 4.3.3 Algorithms for $k \geq 3$.

**Theorem 4.9.** When $k \geq 3$, KG has no polynomial time approximation algorithm with finite approximation ratio unless P=NP.

**Proof.** We show a polynomial reduction of the Perfectly Balanced Graph Partition (PBGP) problem to KG. Because PBGP has no polynomial approximation algorithm with finite approximation ratio unless P=NP [4], our KG has the same hardness.

We review PBGP: given a graph $G = (V, E)$, with weight $w(e)$ on each edge $e$. For an integer $p \geq 2$, a $p$-partition is $p$ disjoint subsets with equal sizes: $V = V_1 \cup \cdots \cup V_p$. We assume $|V|$ is a multiple of $p$ here, so $|V_1| = \cdots = |V_p| = |V|/p$. The decision version of PBGP is that given a positive integer $W$, is there a $p$-partition such that for the cross edges set $E' \subset E$, which have two endpoints in the two different sets $V_i$, i.e., $E' = \{(u_i, v_j) \in E \mid u_i \in V_i, v_j \in V_j, i \neq j\}$, the sum of the weights of edges in $E'$ is less or equal to the given integer $W$, i.e., $\sum_{e \in E'} w(e) \leq W$?

Please refer to the appendix (Sec. A.2) for the details of the proof. Because we could show that PBGP $\leq_P$ KG, and since PGBP has no polynomial time approximation algorithm with finite approximation factor unless P=NP [4], KG has the same hardness.

Because of the intractability of KG, we propose an efficient greedy method to find a $k$-division. The greedy algorithm packs $k$-group one by one, each time starting with an empty set, and keeps adding a new $2$-group $\{p_1, p_2\}$ with smallest OScore to the current $k$-group. If $k$ is an odd number, it randomly picks the last matched pair $p$. The detailed steps is shown in Alg. 1.

At Line 6, the algorithm uses a heaping storing all combinations of pairs in the baseline matching $\{p_1, p_2\}$ with its OScore as the sorted key. Every time we could pop the pair $\{p_1, p_2\}$ with the smallest OScore and add them to the current $k$-group. After each $k$-group is formed, we continue to the next one until a $k$-division is obtained and returned. Line 10-13 greedily add two matched pairs to the current $k$-group $g$, until its size reaches $k$. In total, we form $d$ groups,
as calculated at Line 7. We omit some details for checking used workers and randomly picking the last item (see full version in the appendix, Sec. A.2).

The time complexity of Alg. 1 is dominated by the heap construction, which takes \( O(\varepsilon \log \varepsilon) \), where \( \varepsilon \) is the total number of elements inserted into the heap. We know \( \varepsilon \) is all the combinations of matched pairs, as shown at Line 2-3, so \( \varepsilon = O(n^2) \), where \( n = \max(|W|, |T|) \), the size of the baseline matching. In conclusion, for Alg. 1 has an \( O(n^2 \log n^2) \rightarrow O(n^2 \log n) \) time complexity.

### 4.4 k-HE protocol

The purpose of the previous step Grouping is to divide all worker-task pairs, as shown at Line 2-3, so \( \varepsilon = O(n^2) \), where \( n = \max(|W|, |T|) \), the size of the baseline matching. In conclusion, for Alg. 1 has an \( O(n^2 \log n^2) \rightarrow O(n^2 \log n) \) time complexity.

#### 4.4.1 k-HE protocol

The purpose of the previous step Grouping is to divide all worker-task pairs, as shown at Line 2-3, so \( \varepsilon = O(n^2) \), where \( n = \max(|W|, |T|) \), the size of the baseline matching. In conclusion, for Alg. 1 has an \( O(n^2 \log n^2) \rightarrow O(n^2 \log n) \) time complexity.

#### 4.4.2 k-HE protocol

The purpose of the previous step Grouping is to divide all worker-task pairs, as shown at Line 2-3, so \( \varepsilon = O(n^2) \), where \( n = \max(|W|, |T|) \), the size of the baseline matching. In conclusion, for Alg. 1 has an \( O(n^2 \log n^2) \rightarrow O(n^2 \log n) \) time complexity.

#### 4.4.3 k-HE protocol

The purpose of the previous step Grouping is to divide all worker-task pairs, as shown at Line 2-3, so \( \varepsilon = O(n^2) \), where \( n = \max(|W|, |T|) \), the size of the baseline matching. In conclusion, for Alg. 1 has an \( O(n^2 \log n^2) \rightarrow O(n^2 \log n) \) time complexity.

#### 4.4.4 k-HE protocol

The purpose of the previous step Grouping is to divide all worker-task pairs, as shown at Line 2-3, so \( \varepsilon = O(n^2) \), where \( n = \max(|W|, |T|) \), the size of the baseline matching. In conclusion, for Alg. 1 has an \( O(n^2 \log n^2) \rightarrow O(n^2 \log n) \) time complexity.
5.2 Experimental results

5.2.1 Effectiveness. Overview: \textit{k}-Switch (short as KS in the figures) outperforms other baselines by significant margins, over different privacy levels (Fig. 10), across datasets of different sizes (Fig. 11). For a stricter privacy parameter $\epsilon = 0.4$ on the taxi dataset (shown in Fig. 10(a)), \textit{k}-Switch achieves $5.9\times$ improvement over baseline Oblivious-M, and $1.74\times$ improvement over the competing online method SCGuard.

Details of results. Fig. 10 shows the number of successfully assigned tasks obtained by different methods, over different privacy levels on a dataset of 100 workers vs. 100 tasks. We compare the optimal matching with the Oblivious baseline methods. The optimal matching (OPT) is obtained by using the ground-truth locations, which are not available in the inputs to our PBTA problem. The Oblivious baselines OM and ORR are using only the perturbed locations. At privacy level $\epsilon = 0.4$ (shown in Fig. 10(a)), the gap between OPT and OM is 82. The OPT is about 11 times larger than OM. This validates the motivation of our research: privacy-preserving techniques perturb the locations of workers and tasks, and directly assigning tasks based on perturbed locations is erroneous and prone to sub-optimal assignment. The gap is also observable on the synthetic dataset, shown in Fig. 10(b).

Varying privacy levels: \textit{k}-Switch outperforms all other methods on different privacy levels, assigning 47 tasks for $\epsilon = 0.4$ on taxi data (shown in Fig. 10(a)), achieving $5.9\times$ improvement over OM, which assigns 8 tasks. It also achieves $1.74\times$ improvement over the SCGuard, which assigns 27 tasks. When $\epsilon$ gets larger, the privacy requirement gets less strict, the gap between the OPT and the oblivious OM and ORR gets smaller. For $\epsilon = 2.5$, OM assigns 74 tasks, much closer to the 91 tasks from the OPT solution, as compared to a stricter $\epsilon$. Nevertheless, \textit{k}-Switch delivers strong performance, and assigns 87 tasks, which is close to the OPT.

Varying data size: Fig. 11 shows the number of successfully assigned tasks of different methods, on datasets of different sizes. First, as the number of worker and task increases from 200 to 1000, the OPT result increases from 191 to 999 on taxi dataset (Fig. 11(a)), and 192 to 1000 on the synthetic dataset (Fig. 11(b)). The gap between OPT and OM is consistently large. For the taxi dataset (Fig. 11(a)), the OPT/OM ratio is $191/22 = 8.7$ for $w, t = 200$ and $999/136 = 7.3$ for $w, t = 1000$, respectively. \textit{k}-Switch significantly improves over the baseline OM and the online SCGuard method. It achieves $5.05\times$ improvement over OM for $w, t = 200$, $5.33\times$ for $w, t = 500$, and $5.64\times$ for $w, t = 1000$. As for the comparison with SCGuard, \textit{k}-Switch obtains $1.32\times$, $1.43\times$, and $1.30\times$ improvement for $w, t = 200$, 500, and 1000 respectively. The same behavior is observed on the synthetic dataset (Fig. 11(b)).

Varying parameter $k$: Fig. 12 shows the number of successfully assigned tasks, over different $k$, on datasets of different sizes (Fig. 12(a)) and over different privacy requirements (Fig. 12(b)). As Fig. 12(a) shows, when $k$ increases, the number of successfully assigned tasks increases. For the smallest data size, the utility increases from 47 to 75 tasks, as $k$ increases from 2 to 8. On the other hand, on the data size $w, t = 100$, when we vary privacy parameters, the effect of $k$ is not as significant (Fig. 12(b)).

Varying parameter $\lambda$: We defer the results about varying the system parameter $\lambda$ to the appendix (Sec. B.1). The results verify that $\lambda$ achieves a tradeoff between the utility and efficiency for \textit{k}-Switch, which is consistent with our system design.

5.2.2 Efficiency. Overview. While we expect \textit{k}-Switch to be slower than other methods as the design trades off moderate system overhead with significant utility gain, experimental results show it is only slightly slower than other methods. On our default setting (Fig. 13), 100 workers vs. 100 tasks, it takes only around 2 seconds to run on a laptop machine. When tested on larger sizes dataset (Fig. 14), \textit{k}-Switch shows stable efficient running time, a small constant factor slower than the competing SCGuard. Experiments verify our time complexity analysis of the methods.

Details of results. Fig. 13 shows the running time of different methods (in seconds), over different privacy parameters, on datasets of 100 workers vs. 100 tasks. \textit{k}-Switch obtains around 2 seconds running time on the taxi dataset (Fig. 14(a)), and strictly less than 2 seconds on the synthetic dataset (Fig. 14(b)). The running time is consistent across all privacy levels. When compared with the competing online method SCGuard, \textit{k}-Switch is slightly slower, with a constant factor of 2.
When we vary $\kappa$ we have discussed the most directly related online methods [18, 21] different privacy levels, 100 workers vs. 100 tasks, for different privacy parameters (Fig. 15(b)). As expected, when we report their sensing results, rather than considering our task assignment problem, data publishing has been considered in [28]. Privacy-preserving crowd-sensing is considered in [24], and the focus is to protect the locations of workers when they find that switching tasks between them improves the number of successfully assigned tasks, they swap tasks. Extensive experiments demonstrate that $k$-Switch is both effective and efficient, achieving significant utility gains with reasonable system overhead.

## 7 CONCLUSION

In this work, we target the Privacy-preserving Batch-based Task Assignment (PBA) problem, where both workers and tasks use Geo-1 to perturb their locations before sending them to the untrusted server. We propose a novel solution $k$-Switch, which divides the workers into small groups, and uses a secure computation protocol $k$-HE for inner group communication. If workers inside the groups find that switching tasks between them improves the number of successfully assigned tasks, they swap tasks. Extensive experiments demonstrate that $k$-Switch is both effective and efficient, achieving significant utility gains with reasonable system overhead.

## ACKNOWLEDGMENTS

Lixin Zheng’s work is supported by the National Natural Science Foundation of China No. 62102463 and the Basic and Applied basic Research Project of Guangzhou basic Research Program 202102080401. Peng Cheng’s work is sponsored by the National Natural Science Foundation of China No. 62102149 and Shanghai Pujiang Program 19PJ1403300. Maocheng, Jiachuan and Lei Chen’s work is partially supported by National Key Research and Development Program of China Grant No. 2018AAA0101100, the Hong Kong RGC CRF Project 16209519, CRF Project C6303-18G, C1031-18G, C5026-18G, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC IIT grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, HKUST-NAVER/LINE AI Lab, Didi-HKUST joint research lab, and HKUST-WeBank joint research lab grants. Xuemin Lin’s work is supported by ARC DP200101338. Corresponding author: Peng Cheng.

### 6 RELATED WORK

For privacy-preserving task assignment in spatial crowdsourcing, we have discussed the most directly related online methods [18, 21] in Sec. 1. Here, we include more related works in a broader context of privacy-preserving spatial crowdsourcing.

**Different protection methods.** Encryption-based techniques have been used to compute the exact assignment between workers and tasks [14]. The computational cost of such pure encryption-based techniques is high and prohibitive for real-world applications. Other privacy protection technique, such as cloaking, is used to protect locations of workers [16], but cloaking is considered as a weaker privacy-preserving technique than Geo-I (Sec. 2.2), because its assumption on adversaries’ prior knowledge. There are other related works using Geo-I as the privacy standard [19, 20]. However we adopt a stricter privacy model that the server is untrusted from all workers and tasks, and locations of both parties need to be perturbed before released to any other parties. Similar to our work, the batch-based (offline) setting has also been considered in [23]. However only the workers are protected.

**Other crowdsourcing setting.** There are other related works in the spatial crowdsourcing spectrum [24, 25, 28]. Different from the task assignment problem, data publishing has been considered in [25]. The truthful rather than privacy-preserving task assignment is considered in [28]. Privacy-preserving crowd-sensing is considered in [24], and the focus is to protect the locations of workers when they report their sensing results, rather than considering our task assignment setting, where workers need to move to a specified location of the assigned task, and both locations (tasks and workers) are perturbed with differential privacy.

### Figure 13: Running time (seconds) for different methods, over different privacy levels, 100 workers vs. 100 tasks, $k = 2$.

| Method      | Privacy requirement | OPT     | OM     | ORR    | SCG    | KS     |
|-------------|---------------------|---------|--------|--------|--------|--------|
| eps=0.4     | 0.1                 | 1.6     | 1.4    | 1.3    | 1.3    | 1.3    |
| eps=1.25    | 0.1                 | 1.6     | 1.4    | 1.3    | 1.3    | 1.3    |
| eps=2.5     | 0.1                 | 1.6     | 1.4    | 1.3    | 1.3    | 1.3    |

### Figure 14: Running time (seconds) on different input sizes.

| Data size   | Worker size | k=2       | k=4       | k=6       | k=8       | Groundtruth (OPT) |
|-------------|-------------|-----------|-----------|-----------|-----------|--------------------|
| w,t=100     | 2           | 1.93      | 7.97      | 1.20      | 4.60      | 2.30               |
|             | 4           | 1.70      | 4.51      | 1.15      | 3.90      | 2.25               |
|             | 8           | 1.81      | 5.01      | 1.21      | 4.71      | 2.50               |

### Figure 15: Running time (seconds) of $k$-Switch for different $k$.

Varying data size: We look at the asymptotic growth of the running time across different sizes of datasets in Fig. 14. First, the results verify the quadratic time complexity $O(\lambda k^3 + \lambda n^2 \log n)$, see Sec. 4.5 in terms of $n$, where $n \approx \max(|W|, |T|)$, the input size. When the input size doubles, from $w = 500$ to $w = 1000$, the running time increases about 4 times, from 82.4 seconds to 324 seconds on the taxi dataset (Fig. 14(a)). When compared with SCGuard, SCGuard has the same quadratic time complexity, so it also increases about 4 times, from 30.02 to 120.4 seconds when $w = 500$ increases to $w = 1000$. $k$-Switch is about a 2 times constant factor slower than SCGuard. The same trend is observed on the synthetic dataset (Fig. 14(b)).

Varying parameter $k$ and $\epsilon$: Fig. 15 shows the running time of $k$-Switch, over different $k$, on datasets of different sizes (Fig. 15(a)) and for different privacy parameters (Fig. 15(b)). As expected, when the data size is fixed, the running time of $k$-Switch is larger when $k$ is increased, while the increase is not significant. When we vary the privacy parameter $\epsilon$, the running time of $k$-Switch is stable, across different values of $k$ (Fig. 15(b)).
Algorithm 2: Oblivious-RR

**Input:** Worker set \( W \) and task set \( T \). Perturbed locations \( l_w' \) and \( l_t \) for each \( w \in W \) and \( t \in T \).

**Output:** An assignment (matching) \( M \) between workers and tasks.

1. \( g := \) an empty flow network
2. \( s := \) source node; \( d := \) target (sink) node
3. \( g.addNode(s); g.addNode(d) \)
4. foreach \( w \in W \) do
   - \( e := (s, w, \text{capacity} = 1.0); g.addEdge(e) \)
5. foreach \( t \in T \) do
   - \( e := (t, d, \text{capacity} = 1.0); g.addEdge(e) \)
6. foreach \( w \in W \) do
   - \( \text{prob} := \text{ComputeProb}(l_w', l_t', R_w) \)
   - \( e := (s, w, \text{capacity} = \text{prob}); g.addEdge(e) \)
7. \( f := \text{Ford-Fulkerson}(g) \)
8. \( M := \{ \} \)
9. foreach \( w \in W \) do
   - \( \text{wt}_{\text{sum}} := 0 \)
   - foreach \( t \in T \) do
     - \( \text{wt}[t] := f(w, t) \)
     - \( \text{wt}_{\text{sum}} + := \text{wt}[t] \)
   - \( t_{\text{selected}} := \text{sample } t \in T \text{ with a prob. } \text{wt}[t]/\text{wt}_{\text{sum}} \)
10. \( M.\text{insert}(w, t_{\text{selected}}) \)
11. return \( M \)

Figure 16: An illustrative example of the graph transformation in Alg. 3.

### A TECHNICAL DETAILS

#### A.1 Oblivious-RR

First, for the fractional reachability graph, the weight for an edge \( w, t_j \) is given by \( Pr(d(w, t_j) \leq R_w | d(w', t_j')) \), i.e., the likelihood that the distance between worker \( w \) and task \( t_j \) is indeed smaller than the range \( R_w \) using true locations, given the observed distance between their perturbed locations. We directly adopt analytical approach proposed in [21] (see Sec. IV-B of [21] for details), which uses binomial distribution to approximate the planar Laplace distribution in Geo-I. The calculation is approximate not exact, however it suffices as we only use it to measure the relative high/low likelihood of reachability. We then run the max-flow algorithms to obtain the maximum flow. Note the difference of this step with Figure 4, which obtains the maximum cardinality because of the integral 0 or 1 capacity of edges, here we obtain the flow with the maximum-weight flow. The obtained flow is shown in Fig. 6(a).

The last step is to randomly round the fractional flow to obtain a matching. For each worker \( w_i \), we randomly select a task \( t_j \) with probability proportional to the units of flow sent on the edge (\( w_i, t_j \)). Using Fig. 6(a) as an example, for \( w_3 \), we set the probability of selecting \( t_1 \) as \( 0.1/(0.1 + 0.4 + 0.48) = 0.1/0.98 \approx 0.102 \), selecting \( t_2 \) as \( 0.4/0.98 \approx 0.408 \) and selecting \( t_3 \) as \( 0.48/0.98 \approx 0.49 \). \( t_2 \) has the highest probability of being selected.

In Fig. 6(b), we show a particular matching obtained after the randomized rounding. If we use the true locations shown in Fig. 5 to check the reachability constraint, it turns out that this matching has 3 valid assigned tasks, which equals to the optimal matching.

The details of Oblivious-RR is shown in Algorithm 2. Similar to Oblivious-M, it constructs the flow network by adding the superficial source/sink nodes. Different from Oblivious-M, the edge weight is calculated based on the reachability likelihood at Line 10 (see Sec. IV-B of [21] for how to calculate the likelihood). Line 15-20 execute the randomized rounding, selecting a task for a worker randomly proportional to the units of flow sent on the edge connecting them. The time complexity of Oblivious-RR is the same with Oblivious-M, as it only adds a post randomized rounding which takes \( O(n^2) \) time, with \( n = \max(|W|, |T|) \). Overall, Oblivious-RR runs in \( O(n^2) \) time.

#### A.2 k-Switch

**Extended proof of Theorem 4.7:**

**Proof.** (Continued.) Recall that a matching on a graph is a set of edges without common vertices. The maximum weight matching is a matching with maximum sum of the weight on the edges of the matching.

We show two directions. First, if we have a \( 2 \)-division \( D = \{g_1, \ldots, g_k\} \) of the KGP, it could be mapped to a matching \( m \) on \( M_G \). The matching \( m \) is obtained by only including edges within each \( 2 \)-group \( g_i \in D \), i.e., \( m = \{e = (p_i, p_j) : \exists q \rightarrow p_i, p_j \in g_q\} \). We show the edge set \( m \) is indeed a matching. According to Def. 4.5, \( 2 \)-groups are non-overlapping. This ensure that no edges in \( m \) shares common vertices, as they only include edges within each \( 2 \)-groups, and \( 2 \)-groups are non-overlapping. So edges \( m \) do not share vertices, and thus form a matching.

Then, we show the other direction, if we have a matching \( m = \{e = (p_i, p_j)\} \) on \( M_G \), it could be mapped to a \( k \)-division \( D \) of KGP. We create the \( k \)-division as follows: for each edge \( e = (p_i, p_j) \), we create a \( 2 \)-group \( g = (p_i, p_j) \) and insert it to \( D \). As \( m \) is matching, so edges don’t share common vertices, all created \( 2 \)-groups don’t share any matched-pairs, and the collection \( D \) forms a \( k \)-division.

**Graph transformation:**

Alg. 3 transforms a complete graph to a bipartite graph. See Figure 16 for an example of a graph containing 4 nodes. The basic idea is to copy \( n \) nodes to \( 2n \) nodes, where the 1st node is copied to its copied node (denoted by \( n'_i \)). The original and the copied nodes are put in sets \( L \) and \( R \), respectively. Then we connect every node in set \( L \) with every other node in set \( R \), except for the node copy of itself. Also, we don’t add redundant edges, meaning we only add edges between \( n_2 \) to node \( n'_3 \) and \( n'_4 \) but not to \( n'_1 \), because edge \( (n_2, n'_1) \) is the same as \( (n_1, n'_2) \).

**Extended proof of Theorem 4.9:**
**Algorithm 4: Greedy-Grouping**

**Input:** A baseline matching \( M_0 \). Perturbed locations \( l_w' \) and \( l_t' \) for each \( w \in W \) and \( t \in T \).

**Output:** A k-division \( D \).

1. Initialize a heap \( h \).
2. foreach \( p_i \in M_0 \) do
3.   \( \text{used}[p_i] = \text{False} \)
4.   foreach \( p_j \in M_0 \) do
5.     if \( p_i, w \neq p_j, w \) then
6.       \( \text{oscore} = d(l_{p_i, w}', l_{p_j, w}') + d(l_{p_i, w}', l_{p_j, w}') \)
7.       \( h.\text{insert}((p_i, p_j), \text{oscore}) \) // Push 2-group \( \{p_i, p_j\} \) with its OScore into the heap
8.   end if
9. end foreach
10. \( d := |M_0|/k \) // Calculate how many k-groups to create
11. \( D := \{\} \)
12. for \( i := 0; i < d; i = i + 1 \) do
13.   \( g := \{\} \)
14.   while \( g.\text{size}() < k \) do
15.     if \( \text{size} = k - 1 \) then
16.       \( \text{while True do} \)
17.       \( p := \text{random}(|M_0|) \)
18.       if \( \text{used}[p, w] \) is \text{False} then
19.         \( \text{used}[p] = \text{True} \)
20.         \( g.\text{insert}(p) \)
21.       end if
22.     end while
23.     \( \text{used}[p_1, w] \) is \text{False and used}[p_2, w] is \text{False}
24.     \( h.\text{pop}() \)
25.   end if
26.   \( g.\text{insert}(p_1) \)
27.   \( g.\text{insert}(p_2) \)
28. \( D.\text{insert}(g) \)
29. end for
30. return \( D \)

**Algorithm 3: GraphTransform**

**Input:** Complete graph \( G = (V, E) \)

**Output:** Bipartite graph \( G' \)

1. Initialize an empty graph \( G' \)
2. foreach \( n_i \in V \) do
3.   \( \text{insert} n_i \) to \( G' \)
4.   Create a new node \( n'_i \), insert it to \( G' \)
5. foreach \( n_j \in G \) do
6.   \( e := \text{edge between } n_i \) and \( n_j \)
7.   \( w := w(e) \)
8.   Add edge \( e' = (n_i, n'_j, \text{weight} = w) \) to \( G' \)
9. return \( G' \)

**Proof.** We show that there is a yes instance for the PBGP iff. there is a yes instance for KGP. We first define the decision version of KGP: given a positive integer \( J \), does there exist a k-division \( D = \{g_1, \ldots, g_d\} \), such that score(\( D \)) \leq J ?

For KGP, we construct a graph \( G' = (V', E') \) similar to the proof of Theorem 4.7 when \( k = 2 \). The vertex set \( V' \) corresponds to each matched worker-task pair \( p \) from the baseline matching \( M_0 \). Between every vertex \( p_i \) and \( p_j \) in \( G' \), we add an edge \( e = (p_i, p_j) \). This edge corresponds to a 2-group, with weight \( w(e) = \text{OScore}((p_i, p_j)) \).

If direction: if there is a yes instance to the KGP, we could find a yes instance to the PBGP. Note that for the rest of the proof, we assume the input for the KGP, the baseline matching \( M_0 \) has a size \( |M_0| \) being a multiple of \( k \), for simplicity. The hardness result holds w.l.o.g. because following proof holds on the special case of KGP, so the general version is even harder. Say for a given integer \( J \), we could obtain a k-division \( D = \{g_1, \ldots, g_d\} \), such that score(\( D \)) \leq J . Such k-division corresponds to a p-partition. For each k-group, it contains \( k \) nodes. We have \( p = \lfloor |V'|/k \rfloor \) k-groups in total. The score of the k-division is score(\( D \)) \leq J , and we look closely at what edges are included. Since score(\( D \)) = \( \sum_i \) OScore(\( g_i \)), it sums up all OScore of all the k-groups. For each k-group, OScore(\( g_i \)) = \( \sum_{1 \leq s \leq k} \) OScore(\( \{p_s, p_s\} \)), summing up the edges weight between the nodes within a particular k-group. This k-division corresponds to a p-partition, and we let \( W = J \). The weight sum corresponds to all the edges which are within each partition, which is the k-groups. And because score(\( D \)) \leq J , we know the the sum of weights within each partition in the p-partition is smaller or equal than \( W \).

Only direction: if there is a yes instance to the PBGP problem, then we could obtain a yes instance for the KGP. Given the graph and the p-partition, we know we have \( V = V_1 \cup \cdots \cup V_p, |V_1| = \cdots |V_p| = |V'| \). And for a given positive integer \( W \), for the edges \( E' \subset E \) that have two endpoints in the two different sets \( V_i, E' = \{(v_i, v_j) \in E|v_i \in V_i, v_j \in V_j, i \neq j\} \), the sum of the weights on all such edges is smaller or equal to the given integer \( W \). We search for the graph to be a complete graph by adding edges between nodes that don’t edges, and setting the edge weight to be 0. Since we know we have a yes instance, adding such 0 weight edges to the graph would not increase the cut edges (edges cross different partitions), and the transformed instance would still be a yes instance. We then transform each \( V_i \) to a corresponding k-group \( g_i \), where we set \( k = \lfloor |V'|/p \rfloor \). Each \( g_i \) contains all the nodes in \( V_i \). Since we’ve transformed the graph to be complete graph, each pair of nodes in \( g_i \) has edges between them, and for any edge \( e = (n_i, n_j) \) we define OScore(\( (n_i, n_j) \)) = w(e). So obviously for this constructed k-size \( g_i \), we have the OScore defined, by summing up the weight of all the edges within the group. Finally, now we have a k-division \( D = \{g_i\} \), where each \( g_i \) is transformed from the partition \( V_i \), has size \( k \). We define the score for \( G \) to be score(\( D \)) = \( \sum_i \) OScore(\( g_i \)). Because we know all the cross-partition edges sum, \( W' \) is less or equal to \( W \), so all the in-partition edges sum, which is score(\( D \)) = \( W' \leq W \). Set \( J = W \), we’ve obtained a yes instance to KGP, score(\( D \)) \leq J .

**Details of Greedy-Grouping:**
Algorithm 6: SecureDistanceCalculation

Input: Encrypted location of a task \( E_p k(l_t) \), true location of a worker \( l_w \)

Output: The true distance \( |l_w - l_t| \)

1. \( P_a \) computes \( E_p k(l_w, x - l_t, x) = E_p k(l_w, x) * E_p k(l_t, x)^{N-1} \)
2. \( P_a \) computes \( E_p k(l_w, y - l_t, y) = E_p k(l_w, y) * E_p k(l_t, y)^{N-1} \)
3. \( P_a \) and \( P_b \) uses the SecMul protocol [14] to calculate the square, \( sq_x = E_p k((l_w, x - l_t, x)^2) \), and \( sq_y = E_p k((l_w, y - l_t, y)^2) \)
4. \( P_a \) calculates \( sq_d = E_p k((l_w, x - l_t, x)^2) * E_p k((l_w, y - l_t, y)^2) \)
5. return \( sq_d \)

Algorithm 5: k-HE protocol

Input: A k-group \( g = \{p_1, \ldots, p_k\} \)

Output: The set of modified worker-task matching \( M_g \) for \( g \)

1. Let \( w_1, \ldots, w_k \) be the workers inside k-group \( g \)
2. Let \( t_1, \ldots, t_2 \) be the tasks inside \( g \)
3. Randomly elect two nodes (task or worker) as proxy servers \( P_a \) and \( P_b \)
4. \( P_b \) generates its public key \( pk \) and private key \( sk \)
5. \( P_b \) sends its public key to \( P_a \) and all workers and tasks
6. foreach \( w \) do
7. Worker \( w \) sends their encrypted location \( (E_p k(l_w, x), E_p k(l_w, y)) \) to proxy \( P_a \)
8. foreach \( t \) do
9. Task \( t \) sends their encrypted location \( (E_p k(l_t, x), E_p k(l_t, y)) \) to proxy \( P_a \)
10. foreach \( t \) do
11. \( P_a \) and \( P_b \) use SecureDistanceCalculation (Algorithm 6) to calculate \( E_p k(m) = E_p k(|l_w - l_t|^2) \), \( m \) is sent to \( P_b \)
12. \( P_b \) decrypts message using its private key and store the true distance \( m' = D_{sk}(m) = |l_w - l_t|^2 \)
13. \( P_b \) runs an exact matching algorithm to obtain a new matching \( M_g \)
14. return \( M_g \)

Alg. 4 is the full detailed algorithm. At Line 6, the algorithm uses a heap storing all combinations of pairs in the baseline matching \( \{p_1, p_2\} \) with its OScore as the sorted key. In this way, every time we could pop the pair \( \{p_1, p_2\} \) with the smallest OScore, and if both of them have not been inserted to any k-group, we add them to the current k-group. After each k-group is formed, we continue to the next one until a k-division is obtained and returned. Line 10-28 greedily add two matched pairs to the current k-group \( g \), until its size reaches \( k \). In total, we form \( d \) groups, as calculated at Line 8. The time complexity of Alg. 1 is dominated by the heap construction, which takes \( O(e \log e) \), where \( e \) is the total number of elements inserted to the heap. We know \( e \) is all the combinations of matched pairs, as shown at Line 2 and 4, so \( e = n^2 \), where \( n = \max(|W|, |T|) \), the size of the baseline matching. In conclusion, for Alg. 4 has an \( \mathcal{O}(n^2 \log n^2) \) \( \rightarrow \mathcal{O}(n^2 \log n) \) time complexity.

A.3 Details of k-HE protocol

The detailed k-HE protocol is shown in Alg. 5.

Step 1. two random nodes in the k-group are elected to serve as the proxy servers \( P_a \) and \( P_b \). Note that the nodes could be either a worker or a task. The public keys \( pk \) are generated for \( P_b \), and sent to all parties.

Step 2. all the tasks and workers send the encrypted true location to \( P_a \). The true location for a participant \( v \) (\( v \) is a worker or a task) is represented by a 2-dimensional tuple \( (l_v, x, l_v, y) \), and the encrypted coordinates are \( (E_p k(l_v, x), E_p k(l_v, y)) \).

Step 3. the two proxy servers \( P_a \) and \( P_b \) utilize an interactive communication scheme to compute the true distance between the workers and tasks within the group. The interactive communication scheme is shown in Alg. 6. Most of the steps follow Algorithm 1 in HESI [14]. The true distance is obtained and stored at \( P_b \). The purpose of having a two-server structure is to compute the multiplication of two secret numbers, which needs the plaintext of one of the operands. All the communication between the two proxy servers are encrypted using the public key \( pk \). In the end, \( p_b \) uses its private key to decrypt the true distance between workers/tasks in the group.

Step 4. After obtaining the true distance between workers and tasks, \( P_b \) obtains an exact matching w.r.t. to the reachability graph constructed from the true distance. It returns the matching result to the server. Note that only the updated matching result is returned, the actual distance of the tasks are stored in \( P_b \) only, and the true locations are always strictly protected and encrypted.

B EXPERIMENTS

Real-world dataset. For the real-world dataset, we use the taxi dataset from The dataset contains detailed trajectory samples from the taxi cars running in Xi’an city, Shannxi province of China. On each day, the dataset contains around 30 million samples, with each data sample recording the GIS coordinates (latitude and longitude) of the taxi, the taxi ID, the passenger ID, and the timestamp.

The sampled worker and task locations are in GIS coordinates (latitude and longitude), and we convert it to Cartesian coordinates in meters (in X and Y) using a common technique – Equi-rectangular projection [17], and shift the lower-left boundary point to (0,0). Our points are within the range \([0,8004] \times [0,8259]\).

B.1 Additional results

Figure 17: Effect of parameter \( \lambda \) for k-Switch method on the number of tasks assigned vs. the running time.
Varying parameter $\lambda$. Parameter $\lambda$ controls how many iterations $k$-Switch runs. For each iteration, $k$-Switch improves the number of assigned tasks by running $k$-HE protocol in small groups of size $k$. Thus $\lambda$ controls a tradeoff between the utility of our method vs. the running time. This is verified by the result shown in Fig. 17. When we reduce $\lambda$ from the system default ($\lambda = 20$) to smaller values (e.g., $\lambda = 5, 10$), a moderate decrease of number of assigned tasks is observed, as shown in Fig. 17(a). It is observed consistently across different input sizes. In the meantime, the running time of $k$-Switch is considerably improved, as shown in Fig. 17(b). The running time on data size 500 ($w, t = 500$) is decreased from 82.4 seconds to 18.3 seconds when $\lambda$ is decreased from the system default ($\lambda = 20$) to 5.

B.2 Discussions

As the experiments show, $k$-Switch delivers strong performance in terms of significantly increasing the number of successfully assigned tasks of the baseline matching obtained by various oblivious baselines. Verifying the motivation of the research, $k$-Switch also outperforms the existing online method SCGuard by large margins. Despite more significant utility gain when we increase the value of $k$, we argue that setting $k = 2$ is a cost-effective choice, as it balances the tradeoff between system overhead and utility gain.

In addition, the secure computation we utilize within the small group is to prevent any location leak inside the group. Our experimental results are promising because a small value of $k$ (such as $k = 2$) is sufficient for $k$-Switch to obtain significant utility gain. For real-world adoptions, even if there is malicious attack within the group such that the true locations are disclosed, the potential damage is manageable, given the fact that $k$ is small.