Article

Privacy-Preserving Public Route Planning Based on Passenger Capacity

Xin Zhang [†], Hua Zhang [*,†], Kaixuan Li and Qiaoyan Wen

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; zxby233@bupt.edu.cn (X.Z.); lkx1118@bupt.edu.cn (K.L.); wqy@bupt.edu.cn (Q.W.)
* Correspondence: zhanghua_288@bupt.edu.cn
† These authors contributed equally to this work.

Abstract: Precise route planning needs huge amounts of trajectory data recorded in multimedia devices. The data, including each user's location privacy, are stored as cipher text. The ability to plan routes on an encrypted trajectory database is an urgent necessity. In this paper, in order to plan a public route while protecting privacy, we design a hybrid encrypted random bloom filter (RBF) tree on encrypted databases, named the encrypted random bloom filter (eRBF) tree, which supports pruning and a secure, fast k nearest neighbor search. Based on the encrypted random bloom filter tree and secure computation of distance, we first propose a reverse k nearest neighbor trajectory search on encrypted databases (RkNNToE). It returns all transitions, in which each takes the query trajectory as one of its k nearest neighbor trajectories on the encrypted database. The results can be the indicator of a new route's capacity in route planning. The security of the trajectory and query is proven via the simulation proof technique. When the number of points in the trajectory database and transition database are 1174 and 18,670, respectively, the time cost of an R2NNToE is about 1200 s.

Keywords: public route planning; reverse trajectory query; encrypted trajectory database

MSC: 68P27

Citation: Zhang, X.; Zhang, H.; Li, K.; Wen, Q. Privacy-Preserving Public Route Planning Based on Passenger Capacity. *Mathematics* **2023**, *11*, 1546. https://doi.org/10.3390/math11061546

Academic Editors: Ximeng Liu, Yinbin Miao and Zuobin Ying

Received: 8 March 2023
Revised: 18 March 2023
Accepted: 20 March 2023
Published: 22 March 2023

1. Introduction

Public route planning is used to find a new route that can cover a large area and carry a greater amount of passengers. The operation of a new public route can ease traffic congestion as well as reduce fuel consumption and pollution. Public route planning requires a lot of trajectory data recorded in various GPS-equipped multimedia devices and online location-based services (Bikely, Didi, Twitter, and Facebook) [1]. Since trajectory data include locations, data owners encrypt the trajectory data to preserve their locations' privacy. Public route planning on an encrypted database is necessary.

In a typical scenario of planning a bus route, a passenger's transition includes two points: the source and the destination. The passenger prefers to take the bus, which has stations close to the two points. If a bus company wants to develop a new route (trajectory) that provides services to more passengers, it is necessary to predict the passenger flow of the new route. Note that passengers do not want to leak their location privacy. The new route should not be published until it is applied. Basically, it is a reverse k nearest neighbor trajectory (RkNNT) search on an encrypted trajectory database. The transition data and trajectory data are collected by online location-based service providers; they outsource their encrypted data to the cloud server to release their storage space. In a secure RkNNT search on an encrypted trajectory database, the operations of computing and comparing the distances between different trajectories are frequent, which leads to repeated access to the online location-based service providers. A proxy cloud can represent all the online location-based service providers to cooperate with the server cloud, which can reduce the online computational burden of the online location-based service providers. The details of the two-cloud model are introduced in Section 3.3.

Various kinds of queries on encrypted points are proposed, such as k nearest neighbor (NN) points queries, reverse kNN points queries, range queries, skyline queries and liner range queries. However, all these schemes cannot be applied to an encrypted trajectory query, because the similarity measure of trajectories is based on a more complex aggregation of distances and order between trajectory points, such as dynamic time warping [2], longest common subsequence [3], and edit distance on a real sequence [4]. There are also some schemes study the reverse kNN trajectories query [5,6]. However, they only return the single point, which takes the query trajectory as one of the kNN trajectories. In addition, the locations are not protected, which leaks the locations of users and the points in trajectories. These problems motivate us to investigate the RkNNT search on the encrypted databases.

There are two challenges to search the RkNNT on the encrypted databases. One is to reduce the search space, since computation on large encrypted data is time-consuming. The other is to search on a certain space without leaking the location's privacy. To overcome these two challenges, our main contributions are as follows:

- In this paper, we first design a hybrid tree, eRBFtree. It divides the search space into subspaces according to the distribution of trajectory points. The division of the subspace is according to the distribution of transition points. The eRBFtree supports spatial pruning and fast kNN search on ciphertext.
- We propose a reverse kNN trajectory search on the encrypted database, RkNNToE. We use eRBFtree to prune the space of encrypted transitions. Then, we give a distance list (DList), which helps to refine the transitions and reduce the times of the kNN search. To ensure the correctness of results, we apply the fast kNN search for every transition as a result.
- Theoretical analysis proves that clouds and users cannot know the locations of data and the distance between two locations at the same time. The experiment results confirm that our scheme is practicable in the GeoLife project in Beijing and the bus lines dataset in Beijing.

2. Related Work

In this section, we present an overview of the existing protocols in terms of trajectory search on plain text [7] and secure RkNN search [8], which are related to our work in this paper. The comparison between related schemes and RkNNToE is listed in Table 1. Note that a trajectory can degrade into a point, so the search method in RkNNT can deal with the RkNNP search, and a two-type database can degrade into a one-type database.

Table 1. Comparison with related works.

Schemes	**Plaintext**				**Ciphertext**	
	[5,6]	**[9]**	**[10]**	**[11]**	**[8]**	**RkNNToE**
Search Type	RkNNT	RkNNT	RkNNT	RkNNP	RkNNS	RkNNT
Query Type	T	T	P	P	S	T
Result Type	P	T	T	P	S	T
Database Type	P and T	T and T	P and T	P	S	T and T

P: point; T: trajectory; S: set.

RkNNT Search. In [12,13], an RkNN points search was studied, which is the foundation of RkNNT search. Refs. [5,6] investigated the problem to find the single points—that is, the kNN points—for the query trajectory. In 2018, Wang et al. [9] proposed an RkNN trajectory search, which studies transitions with multiple points. It does not include any semantic information [10]. In [14,15], the reverse spatial–keyword nearest neighbor queries were studied. Pan et al. [10] introduced the geo-textual object sequences to achieve an

RkNN semantic trajectories search. None of the above schemes focus on the privacy of both the query and data.

Privacy-Preserving RkNN Search. In [16], the private information retrieval was used to protect the query to achieve the privacy-preserving RkNN search. It does not protect the database stored in the cloud [17]. Li et al. [17] designed a reference-locked order-preserving based RNN query, which protects the database, but it is only used for two-dimensional data. In [11], RkNN over-encrypted multi-dimensional data were proposed, which only support point data and cannot support trajectory data. In 2023, Zheng et al. [8] proposed a privacy-preserving set reverse kNN query, which is not suitable for the two-type trajectory database.

3. Problem Formulation

The notations are shown in Table 2.

Table 2. Notations.

Notation	Definition
dist(a,b)	The distance between a and b
DB_p	The database of all points
DB_τ	The database of points in all trajectories
DB_o	The database of points in all transitions
S_τ, S_{can}	The set of trajectories and the set of candidate transitions
S_{ref}, S_{res}	The set of refined transitions and the set of results
$node(\cdot)$	The node with identity $(\cdot)$
loc	The vector of location
N_τ	The max number of trajectory points in a leaf node of the father tree
N_o	The max number of transition points in a leaf node of the child tree
$i \in (a, b)$	$i \in (a, \ldots, b)$

3.1. RkNNT Problem and Definitions

The RkNNT on the plain-text database is introduced in [9]. In this paper, we follow their definitions.

Definition 1. *(Transition) A transition of an object $O = (s, d)$ is a pair of points, describing the motive object's source and destination. $\mathcal{D}_o$ is the set of transitions.*

Definition 2. *(Trajectory) A trajectory (route) τ of length l is a sequence of points $< p_1, p_2, \ldots, p_{N_p} >$, where N_p is the number of points in the trajectory, and $\mathcal{D}_\tau$ is the set of trajectories.*

Definition 3. *(Point-to-trajectory distance) The distance between a point p_i and a trajectory τ_j is defined as:*

$$Dist(p_i, \tau_j) = \max_{p_j \in \tau_j} dist(p_i, p_j) \tag{1}$$

Definition 4. *(RkNNT) Given a transition set $\mathcal{D}_o$, a trajectory set $\mathcal{D}_\tau$ and a query trajectory Q, RkNNT(Q) returns all the transitions in a set $\mathcal{D}_1 \in \mathcal{D}_o$. For each $O = (s, d) \in \mathcal{D}_1$, all trajectories $\tau \in \mathcal{D}_\tau$ that meet $Dist(s, \tau) \leq Dist(s, Q)$ and $Dist(d, \tau) \leq Dist(d, Q)$ are stored in a set $\mathcal{D}_2$, whose size less is than k.*

3.2. Basic Security Primitives

3.2.1. CKKS Encryption

CKKS encryption [18] is a fully homomorphic encryption. It can directly encrypt a vector and support calculating the inner product on cipher text. In this paper, $CKKS_{enc}(\cdot)$, $CKKS_{dec}(\cdot)$,$CKKS_{sub}(\cdot,\cdot)$ and $CKKS_{dot}(\cdot,\cdot)$ represent the operation of encryption, decryption, subtraction and inner product, respectively. If $CKKS_{enc}(v_1) = c_1$, $CKKS_{enc}(v_2) = c_2$, $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$, then $CKKS_{dec}(CKKS_{dot}(c_1, c_2)) = v_1 \cdot v_2$, $CKKS_{dec}(CKKS_{sub}(c_1, c_2)) = (x_1 - x_2, y_1 - y_2)$ and $CKKS_{dec}(CKKS_{dot}(CKKS_{sub}(c_1, c_2), CKKS_{sub}(c_1, c_2))) = (x_1 - x_2)^2 + (y_1 - y_2)^2$. In this paper, we use the above operations to obtain the distance of two points and denote a new operation as $CKKS_{dis^2}(c_1, c_2) = CKKS_{dot}(CKKS_{sub}(c_1, c_2), CKKS_{sub}(c_1, c_2))$.

3.2.2. Security *k*NN

In our algorithm, a secure *k*NN point search is based on the Fast and Secure kNN query (FSknn [19]). In this paper, we will briefly give the main changes compared to the FSknn.

Index-building. In this phase, a data owner (DO) firstly random generates two vectors $v_1 \perp v_2$. The method of computing every point's prefix families is the same as it in FSknn. However, in this paper, the DO treats all prefixes of all points in subspace of a node as keywords kw to embed in one RBF rather than all prefixes of a point. As shown in Figure 1, an empty RBF is initialized as a two-row and m-column random binary array. The two elements in the same column are different. $RB[i][j]$ is the element in the i-th row and j-th column of RBF. For every keyword, the DO sets $RBF[H(h(h_k(kw)) \oplus r_k)][h_k(kw)] = 1$ and $RBF[1 - H(h(h_k(kw)) \oplus r_k)][h_k(kw)] = 0$, where $h(\cdot) = HMAC(\cdot)mod2, h_k(\cdot) = HMAC(\cdot)$, $H(\cdot) = SHA256(\cdot)mod2$ and k is the number of hash functions for RBF. Every RBF point is a node rather than a point. An example of inserting a keyword is shown in Figure 1. An RBF tree is generated based on $RBF_p[H(h(h_l(kw)) \oplus r_p)][i] = RBF_l[H(h(h_l(kw)) \oplus r_l)][i] \vee RBF_r[H(h(h_r(kw)) \oplus r_r)][i]$, where RBF_p is the parent RBF of child $RBF_i, i \in (1, 4)$. An example of constructing an RBF tree is shown in Figure 2.

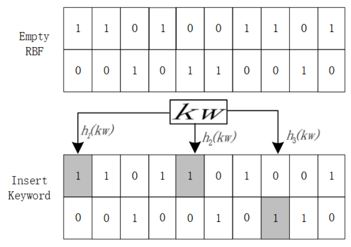

Figure 1. Inserting a keyword on an empty RBF.

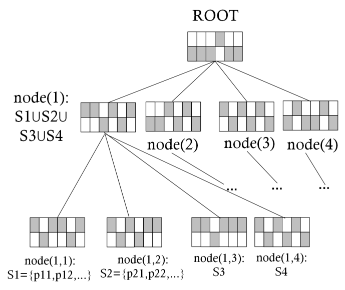

Figure 2. Index structure: RBF tree.

Token generation. When a data user (DU) wants to find the *k*NN in the database, the DU needs to generate *k* pairs of hashes and locations that serve as the search token following the same method in FSknn. However, when the token is generated by the DO, it only needs to generate the token based on one radius dis_{ref} rather than L radiuses.

Query processing. The method of the cloud is the same as it in FSknn. However, the stop condition is to find kNN trajectories in all query points' kNN points set rather than to find more than kNN points for every query point.

Post-processing. If there are not kNN trajectories in all query points' kNN points set, the DU needs to expand the search radius and repeat search kNN points following the same method in FSknn. However, if the token is generated by the DO, it does not need to expand the search radius or repeat search for kNN points.

3.3. The System and Threat Models

As shown in Figure 3, there are four entities: a data owner (DO), two clouds ($cloud_1$ and $cloud_2$) and a data user (DU). The details are described as follows.

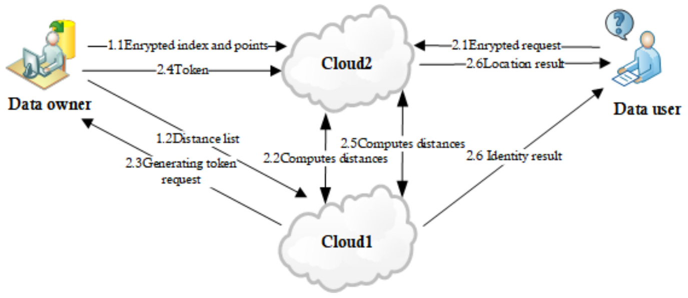

Figure 3. The model of RkNNToE search.

The DO is a data owner. The data include the transition data and the trajectory data. The DO wants to update the encrypted trajectory data and transition data to $cloud_2$ to release the storage space.

The DU is a user who wants to process the RkNNT search on the database stored in $cloud_2$. The DU sends a query to trigger the service; the query includes the encrypted information of the data user's trajectory.

$cloud_1$ (proxy cloud) is the proxy of the DU and DO, which is responsible for directing $cloud_2$ to filter and refine the transitions, and calling the DU to construct the token for every point in the refined transitions.

$cloud_2$ (server cloud) provides storage space for data owners. $cloud_2$ is responsible for searching nearest neighbor points for every point in a query trajectory and refined transitions, computing the distance between points or points and nodes with the $cloud_1$'s help, and sending the encrypted transition points to the DU.

Overview: As shown in Figure 3, the DO sends the index and encrypted points to $cloud_2$ and a distance list (DList) to $cloud_1$ to complete data outsourcing. If the DU wants to conduct an RkNN trajectory search, he sends the encrypted request to $cloud_2$. $cloud_2$ cooperates with $cloud_1$ to prune and refine transitions that cannot be the RkNN transition of the query trajectory. $cloud_1$ obtains the refined transitions and sends a request for NN points token for every point in refined transitions to the DO. The DO generates and sends the tokens to $cloud_2$. $cloud_2$ cooperates with $cloud_1$ to find the NN trajectories of all refined transitions based on the NN points. $cloud_1$ obtains the transitions that take the query trajectory as one of the kNN trajectories and returns the results to the DU.

3.4. Secure Requirements for MTS

Our scheme is under the assumptions that two clouds follow the processing of search and cannot actively attack the system or collude with each other (honest-but-curious). The DO and DU cannot collude with any cloud, but they can be a malicious attacker. Note that we mainly focus on the location privacy of points. The identity is on plain text.

Data Security. The location of every point in the transition and trajectory should not be learned by both clouds. An attacker cannot know the points' locations in the encrypted database.

Index Security. The index is secure, which means that $cloud_2$ cannot know the specific point pointed by every leaf node of the index, and every node cannot reveal the location of both trajectories and transitions.

Query Security. Both the encrypted requests cannot reveal the location of every point in the query trajectory. Both clouds cannot know the specific location.

4. The Proposed Scheme

In this section, first, we generalize the main idea of the search. However, all information of the index is not protected, and the trajectories and transitions are not encrypted. Then, we proposed a secure scheme with encrypted index and encrypted data, which should be processed in a two-cloud model. It can satisfy the secure requirements and counter-threat model.

4.1. Main Idea of RkNNT Search

The reverse trajectories searching are divided into four steps: building a hybrid quad tree, generating a filter set and pruning transition, refining transitions and returning results. The whole processing is shown in Algorithm 1.

Algorithm 1: *Reverse Trajectory Search* (Q, DB_p)

```
    Input: $Q$: query, $DB_p$: all points in the database $DB_p = DB_\tau \cup DB_o$
    Output: $RkNN(Q)$: The reverse kNN trajectories for $Q$
1   $DB_p \rightarrow$ hybrid Quadtree
2   for all $q_i \in Q$ do
3       $k$NN$(q_i, DB_\tau) \rightarrow Table$
4   FilterSet$(Table) \rightarrow S_\tau$
5   PruneTransition$(S_\tau, DB_o) \rightarrow S_{can}$
6   RefineTransition$(S_{can}, DList) \rightarrow S_{ref}$
7   for all $O = \{s, d\} \in S_{ref}$ do
8       $k$NN$(s, DB_\tau) \rightarrow S_s$ $k$NN$(d, DB_\tau) \rightarrow S_d$ $k$NNTrajectorySet$(S_s, S_d) \rightarrow S_{\tau'}$
          CompareDistance$(dis(\tau_{k-th}, O), dis(Q, O)) \rightarrow S_{res}$
9   return $S_{res}$
```

4.1.1. Building Hybrid Quad Tree

On the plain-text trajectory database, we build a hybrid quad tree base on quad tree [20] in DB_p. DB_p includes all the points in DB_τ and DB_o. The space in a node is partitioned into four equal subspaces. The subspace is stored in the child node. The partitioning will not be stopped until there are less than n points in the subspace. First, the partitioning is based on DB_τ, it will not be stopped until there are less than N_τ points in the subspace. The quad tree in this phase is called the father tree. The trajectory points are stored in every leaf node of the father tree. Then, every subspace in the leaf node of the father tree is partitioned. The partitioning is based on all points in this subspace; it will not be stopped until there is less than N_o points in one leaf node. The quad tree takes the leaf node of the father tree, as its root node is called the child tree. Figure 4 shows the structure of a hybrid quad tree. The bold tree is the father tree. The others are child trees. Every non-leaf node of the hybrid quad tree stores the location vectors of four vertexes. Every leaf node stores the identities and location vectors of points in this leaf node. This is shown in line 1 of Algorithm 1.

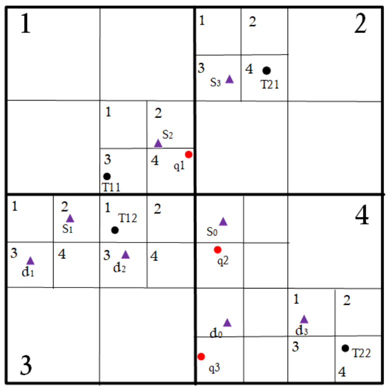

Figure 4. The partition for all points.

4.1.2. Generating Filter Points and Pruning Transitions

If a reverse trajectory search is needed, we find the NN trajectory points for every query trajectory point and construct a table. In Figure 5, the NN points of query points (q_1, q_2, q_3) are in $node(1)$, $node(2)$, $node(3)$ and $node(4)$. Then, we find the trajectory, which has more than two points in the table, such as $T1$. All points in these trajectories are called filter points. In Figure 6, the filter points are $T11$ and $T12$. We form a polyline based on perpendicular bisectors between the points from one trajectory and the query points. The polyline divides the space into two subspaces. If one node is intersected by the polyline, then we check whether the child node meets the above condition. Then, $node(1)$ and the $node(3)$ are intersected by the polyline in Figure 5. Its child node needs to be checked. If the node is the leaf node of the child tree, we list all the transitions' identity and compute the distance between the transition points and the filter points. If there are more than k trajectories closer to the transition than the query trajectory, the transition is pruned. In Figure 6, leaf $node(3,2,3)$ is intersected by the polyline, and we compare the distance $dist(O_2, Q)$ with the distance $dist(O_2, T1)$. Since $dist(O_2, Q) > dist(O_2, T1)$, transition O_2 can be pruned. If one node is in the subspaces of two filter points with one trajectory identity, the node is closer to the trajectory than the query trajectory. If there are more than k polylines that make one node meet the above condition, there are more than k trajectories closer to the node than to the query trajectory. All transitions in these nodes are closer to the k trajectories than to the query trajectory. All transitions in these nodes can be pruned. In Figure 5, the $node(3,1)$ is in the subspace of $T11$ and $T12$, all points in $node(3,1)$ are closer to trajectory $T1$ than to query trajectory Q. Since transition $O_1 = (s_1, d_1)$ is in $node(3,1)$, it can be pruned. All the rest of the transitions are called candidate transitions. The candidate transitions in Figure 5 are O_0 and O_3.

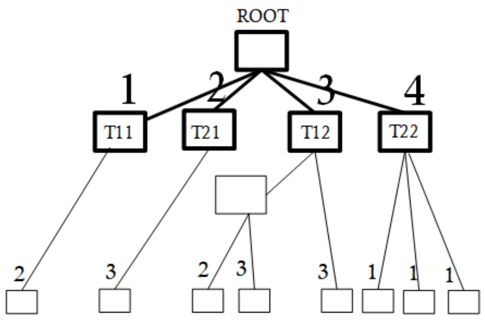

Figure 5. The quad tree structure for all points.

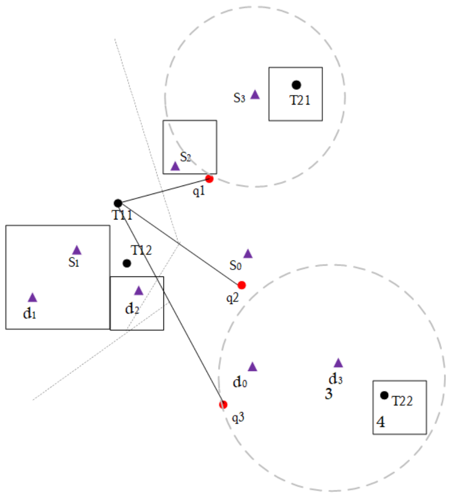

Figure 6. The example of RkNN search.

4.1.3. Refining Transitions and Returning Results

For every candidate transition, we compute the distance between every point in transition and the query trajectory. We check the nodes in the quad tree by using a circle, of which the radius is the distance and the center point is the transition point. For the nodes in the circles of one transition, we record the identities of trajectories in these nodes. For the nodes that intersect with the circle, the child node needs to be checked further. If the node is a leaf node, we compute the distance between every transition point and trajectory in the leaf node. We record the identities of trajectories which are closer to the transition than the query trajectory. If the total number of recorded identities is more than k, then the candidate transition is deleted. In Figure 6, the circles of point s_3 and d_3 are drawn. The nodes $(2,1,4)$ and $(4,4,1)$ are in the circle, respectively. The trajectory $T2$ is closer to the transition O_3. It can be deleted in the candidate transitions. The rest of the candidate transitions are called refined transitions. For every point of the refined transitions, we find the NN points in the quad tree and check whether there are two points of query trajectory in it. If two points of the query trajectory are in the NN points of one transition, it is inserted in the set $RkNN(Q)$. The $RkNN(Q)$ is the search results. In Figure 5, the NN point for the point s_0 is q_2 and the NN point for the point d_0 is q_3. The $RkNNT(Q)$ in Figure 5 is O_1.

4.2. Reverse Search on Encrypted Trajectory Database

In this section, the points of transitions and trajectories are encrypted, and the hybrid quad tree is replaced by an encrypted RBF tree (eRBFtree) and the distance list (DList). This section is consists of four phases: setup, eRBFtree building, query encryption and search.

4.2.1. Setup

The data owner generates the parameters of CKKS and RBF tree as shown in Section 3.2. It encrypted all the location vectors of points in database DB_p. For a point with identity ID and location loc, its item is $\{ID, CKKS_{enc}(loc)\}$. The $cloud_2$ generates its private key sk_2 and public key pk_2; it publishes the public key pk_2 to the DO and DU.

4.2.2. eRBFtree and DList Building

As shown in Figure 7, building an eRBFtree includes two steps. The first step is building the RBF tree in the database DB_τ and the partitioning of space is the same to the partitioning of the father tree in Section 4.1.1. Every leaf node of the RBF tree stores the encrypted items of trajectory points. Every non-leaf node stores an RBF and four encrypted points $\{CKKS_{enc}(V_1), \ldots, CKKS_{enc}(V_4)\}$, where $V_i, i \in (1,4)$ is the four vertices of the node. The second step is building the child trees in the database DB_o. Every leaf node of the child tree stores encrypted items of transition points. Every non-leaf node stores four encrypted points $\{CKKS_{enc}(V_1), \ldots, CKKS_{enc}(V_4)\}$. The DList is a table, in which every row records $(ID_o, p) : \{dis_1, SID_\tau^1\}, \{dis_2, SID_\tau^2\}, \ldots$. The keywords (ID_o, p) are the identity of transition and one point in the transition. The value dis_i is the maximum distance from the point p to its nearby nodes. The value SID_τ^i is the set of trajectories' identities in these nodes. The values are listed in increasing order by the dis_i. The eRBFtree and the DList are constructed by the data owner. The DO encrypts the eRBFtree with all the items by the public key pk_2 and sends the cipher text to $cloud_2$. The DO sends the DList and the secret key of CKKS to $cloud_1$.

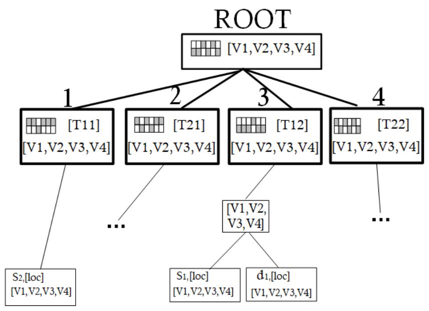

Figure 7. The structure of eRBFtree. $[\cdot]$ is the encryption of $\cdot$.

4.2.3. Query Encryption

The query includes tokens and items for points $q_j, j \in (1, N_p)$ in the query trajectory. N_P is the number of points in the query trajectory. The token $Token(q_j)$ is for a secure kNN search in eRBFtree, which is constructed as shown in Section 3.2.2. The center point is the point of the query trajectory, and the search radius is set by the DO. The item $\{CKKS_{enc}(q_j)\}$ is the encrypted location vector of the point q_j. The query $Q = \{(Token(q_1), CKKS_{enc}(q_1)), \ldots, (Token(q_{N_p}), CKKS_{enc}(q_{N_p}))\}$ is encrypted by the public key of $cloud_2$; then, it is sent to $cloud_2$ to start a reverse search.

4.2.4. Search

In this phase, $cloud_2$ decrypts the query with the private key sk_2. Then, $cloud_2$ uses the tokens $Token(q_j), j \in (1, N_p)$ to search the eRBF tree, obtains the NN trajectory points for every point in the query trajectory, checks the identities of points and constructs the filter set. The item in the filter set is $\{ID_\tau, CKKS_{enc}(loc_1), CKKS_{enc}(loc_2), \ldots, CKKS_{enc}(loc_{N_p})\}$, where $CKKS_{enc}(loc_1), CKKS_{enc}(loc_2), \ldots, CKKS_{enc}(loc_{N_p})$ are NN points of N_p query points. They have the same trajectory identity ID_τ. $Cloud_2$ computes distances between every vertex in the node and the filter points by $DIS_1 = CKKS_{dis^2}(CKKS_{enc}(loc_i), CKKS_{enc}(V_j)), i \in (1,2), j \in (1,4)$. Then, $cloud_2$ computes the distance between every vertex in the node and the query trajectory by $DIS_2 = CKKS_{dis^2}(CKKS_{enc}(q_i), CKKS_{enc}(V_j)), i \in (1, N_p), j \in (1,4)$. Afterwards, $cloud_2$ sends DIS_1, DIS_2 to the $cloud_1$. $Cloud_1$ decrypts them and obtains the distance between every vertex and the filter points $dist(loc_i, V_j), i \in (1,2), j \in (1,4)$ and the distance between every vertex in node and the query trajectory $dist(q_i, V_j), i \in (1, N_p), j \in (1,4)$. The process is from the root node to the leaf node, using the pruning transition in Section 4.1.2. If one node is filtered, $cloud_1$ notifies $cloud_2$. Then, $cloud_2$ stops computing the distance of its child node. If the node is a leaf node, $cloud_2$ and $cloud_1$ compute the distance between every transition point in the node and the query trajectory $dist(loc_i, q_j), i \in (1,2), j \in (1, N_p)$. After filtering the transitions, $cloud_2$ cooperates with $cloud_1$ to compute the distance between the candidate transitions and the query trajectory. The identities of transitions and the cipher text of distance are sent to $cloud_1$. Then, $cloud_1$ decrypts the cipher text and obtains the distance $d = dist(loc_i^{can}, q_j), i \in (1,2), j \in (1, N_p)$. For every candidate point $loc_i^{can}, i \in (1,2)$ in one transition, $cloud_1$ refers to the DList, locates the row of keyword loc_i^{can} and finds the maximum values for dis_{h_i} meet $dis_{h_i} \leq min\{dist(loc_i^{can}, q_j), j \in (1, N_p)\}$. Then, $cloud_1$ counts the number of trajectories, of which two points come from two sets $SID_\tau^{h_1}$ and $SID_\tau^{h_2}$. If the number of the trajectories is more than k, the transition $(loc_1^{can}, loc_2^{can})$ can be pruned. Then, $cloud_1$ sends the identities of refined transitions S_{ref} to $cloud_2$. For every refined transition (s,d) with identity in S_{ref}, $cloud_2$ sends the identity and a distance $dis_{ref} = dist(s,Q) + dist(d,Q)$ to the DO. Then, $cloud_2$ sends the encrypted transition $(CKKS_{enc}(loc_s), CKKS_{enc}(loc_d))$ points to the DO. The tokens $Token_{ref}$ for every point in the refined transition are constructed after the DO obtains the request $\{ID_O, dis_{ref}\}, ID_O \in S_{ref}$ from $cloud_1$ and decrypts $(CKKS_{enc}(loc_s), CKKS_{enc}(loc_d))$. The DO constructs two tokens for every transition, as shown in Section 3.2.2. The center points are the points of location loc_s and loc_d, respectively. The radius is dis_{ref}. The DO sends the set of tokens $Token_{ref}$ to $cloud_2$. $Cloud_2$ searches the NN points and checks if there is less than k trajectories in the NN points. If there are, the transition is one of the reverse k transitions for the query trajectory. Otherwise, $cloud_2$ computes the distance between trajectories and transitions with the help of $cloud_1$. $Cloud_1$ compares the distance between the trajectory and transitions as wel as between the query trajectory and transitions. If more than k trajectories are closer to one transition, the transition is deleted. The rest of the refined transitions are the results. Then, $cloud_1$ returns the identities to the DU and $cloud_2$ returns the encrypted locations to the DU.

5. Theoretical Analysis

5.1. Correctness Analysis

In this section, we will discuss the returned results, which are all reverse transitions for the query trajectory. The discussion is divided into three steps.

(1) In the first step, we find the filter set S_τ and prune the $O = (s,d) \in DB_o$ so that $\exists \mathcal{D}_\tau = \{\tau_1, \dots, \tau_k\}$ such that $dist(s,\tau_i) < dist(s,Q)$ and $dist(d,\tau_i) < dist(d,Q)$, $i \in (1,\dots,k)$. According to Definition 4, the transition cannot be in RkNNT(Q). We call the transition that is not in RkNNT(Q) a negative transition and the transition that is in RkNNT(Q) a positive transition. In this step, we only prune a part of the negative transitions. There are also many negative transitions in set S_{can}.

(2) In the second step, we use the candidate set S_{can} and DList to delete the $O = (s,d), s \in S_{can}$ or $d \in S_{can}$ so that there exists $\{dis_s < dist(s,Q), SID^s_\tau\}$ in the row of point s, $\{dis_d < dist(d,Q), SID^d_\tau\}$ in the row of point d and $SID^s_\tau \cap SID^d_\tau$ has more than k trajectory identities. It also means that $\exists \mathcal{D}_{\tau'} = \{\tau'_1, \dots, \tau'_k\} \subset (SID^s_\tau \cap SID^d_\tau)$ such that $dist(s,\tau_i) < dist(s,Q)$ and $dist(d,\tau_i) < dist(d,Q)$, $i \in (1',\dots,k')$. In this step, we also delete a part of negative transitions. It is unclear whether there are any negative transitions in set S_{ref}.

(3) In the third step, we know that if a transition takes the query trajectory as one of its kNN trajectories, the transition must be the RkNNT of the query trajectory. For every transition $O = (s,d) \in S_{ref}$, we find all trajectory points with distance to s or d less than $dis_{ref} = dist(s,Q) + dist(d,Q)$. If a trajectory τ has only a point with distance to s or d less than dis_{ref}, then $dist(s,\tau) + dist(d,\tau) > dis_{ref}$. If a trajectory τ has no point with distance to s or d less than dis_{ref}, then $dist(s,\tau') + dist(d,\tau') > 2dis_{ref}$. So if only a trajectory has one point with distance to s less than dis_{ref} and the other one point has distance to d less than dis_{ref}, it is possibly closer to the transition $O = (s,d)$ than the query trajectory Q. For every transition $O = (s,d) \in S_{ref}$, we list all NN trajectories τ_i meets $dist(s,\tau_i) + dist(d,\tau_i) \leq 2dis_{ref}$ and check the size of $\mathcal{D}_\tau = \{\tau_1, \dots, \tau_j\}$ such that $dist(s,\tau_i) + dist(d,\tau_i) < dist(s,Q) + dist(d,Q)$, $i \in (1,\dots,j)$. If the size of $\mathcal{D}_\tau$ is not more than k, the transition must be the positive transitions; otherwise, the transition must be the negative transition.

5.2. Security Definitions and Analysis

The two-clouds model is honest-but-curious, and the RkNNToE is processed in two phases. The definition of leakage functions [21] of two phases and the formal proof are proposed. It shows that RkNNToE is secure in an honest-but-curious clouds model.

Definition 5. *In an honest-but-curious clouds model, there are two participants $\mathcal{C}i, i \in (1,2)$ in a protocol $\mathcal{P}$. For $\mathcal{C}_i$, f_i and O_i are the execution function and its output, while $view_i$ is the view* *during an execution of $\mathcal{P}$. The protocol $\mathcal{P}$ is secure against a probabilistic polynomial time (PPT) honest-but-curious adversary if there exist simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ such that:*

$$(\mathcal{S}_1(f_1, \mathcal{L}_1), f_2) \equiv (view_1, O_2) \tag{2}$$

$$(f_1, \mathcal{S}_2(f_2, \mathcal{L}_2)) \equiv (O_1, view_2) \tag{3}$$

where $\equiv$ means computational indistinguishability.

$\mathcal{L}_i^j$ is the leakage function of cloud $i \in (1,2)$ in phase $j \in \{setup, search\}$. Given a collection of points DB_p from the DO and a query trajectory Q from the DU,

$$\mathcal{L}_1^{setup}(DB_p) = \{DL, [\mathbb{EI}, [id, p]]\}$$

$$\mathcal{L}_2^{setup}(DB_p) = \{\mathbb{EI}, (OID, [loc])_i, (TID, [loc])_j, |DB_p|, |DB_\tau|, |DB_O|\}$$

$$\mathcal{L}_1^{search}(DB_p, Q) = \{\mathbb{D}(Q), DL, S_{can}, S_{ref}\}$$

$$\mathcal{L}_2^{search}(DB_p, Q) = \{Token_i, Token_j, |Q|, |S_{ref}|, |S_{can}|, \mathbb{S}(Q), \mathbb{A}(Q), (OID, [loc])_i, (TID, [loc])_j\},$$

where DL is the distance list, EI is the eRBF tree, id is the identity of point p, $[\cdot]$ is the cipher text of $\cdot$, $|\cdot|$ is the size of $\cdot$, OID_i is the identity of transition i and TID_j is the identity of trajectory j.

Definition 6. *(Search Pattern $\mathbb{S}$) The search pattern leakage reveals whether the keywords in the token of every query point have appeared before.*

Definition 7. *(Access Pattern $\mathbb{A}$) Given a search query Q, the access pattern is defined as the identifier of trajectory points in the nearest neighbor of query points.*

Definition 8. *(Distance Pattern $\mathbb{D}$) Given a search query Q, $\mathbb{D}(Q) = dist(p_i, q_j), q_j \in Q, p_i \in S_{can}$. Informally, this part of leakage can be derived from the query, $\mathbb{D}$ leaks the distances between the points in candidate transitions and query points.*

Theorem 1. *Under the permitted leakage functions $\mathcal{L}_1^{Setup}$, $\mathcal{L}_2^{Setup}$, $\mathcal{L}_1^{Search}$ and $\mathcal{L}_2^{Search}$, if CKKS and the FSknn [19] are secure in the two honest-but-curious clouds model, then RkNNToE is secure in the two honest-but-curious clouds model.*

Proof. We introduce the leakage function to Definition 5 and prove that for any PPT adversary, there exist simulators S_1 and S_2 such that:

$$(\mathcal{S}_1(f_1, \mathcal{L}_1^{setup}), f_2) \equiv (view_1, O_2) \tag{4}$$

$$(\mathcal{S}_1(f_1, \mathcal{L}_1^{search}), f_2) \equiv (view_1, O_2) \tag{5}$$

[Simulating Setup] Given $\mathcal{L}_1^{setup}(DB_p) = \{DL, [\mathbb{EI}, [id, p]]\}$, $\mathcal{S}_1$ randomly generates a message as the plain text m and encrypts it by using a CPA-secure encryption to obtain $[m]$. $\mathcal{S}_1$ randomly generates the identity of trajectories and transitions. The number of these trajectories is the same as the one listed in DL. $\mathcal{S}_1$ randomly generates many increasing arrays to represent the distance between the transition points and vertices of each node. Since the PPT adversary does not know the real distribution of points, and the encryption in the above simulation is secure, a PPT adversary cannot distinguish between the simulated view and the real view.

[Simulating Search] Given $\mathcal{L}_1^{search}(DB_p, Q) = \{\mathbb{D}(Q), DL, S_{can}, S_{ref}\}$, $\mathcal{S}_1$ knows the identities of transitions that are deleted in the phase of refining transitions $S_{del} = S_{can} - S_{ref}$. From $\mathbb{D}(Q)$, $\mathcal{S}_1$ knows the distance between the point in S_{can} and query points. In the simulated DL', if a transition is in S_{del}, it must have kNN trajectories closer than the query.

A PPT does not know the locations of every point; it only knows the distance and the identities of deleted transitions. It cannot distinguish between the simulated DL' and the real DL.

$$(f_1, \mathcal{S}_2(f_2, \mathcal{L}_2^{setup})) \equiv (O_1, view_2), \tag{6}$$

$$(f_1, \mathcal{S}_2(f_2, \mathcal{L}_2^{search})) \equiv (O_1, view_2) \tag{7}$$

[Simulating Setup] Given $\mathcal{L}_2^{setup}(DB_p) = \{\mathbb{EI}, (OID, [loc])_i, (TID, [loc])_j, |DB_p|, |DB_\tau|, |DB_O|\}$, $\mathcal{S}_2$ randomly chooses $|DB_p|$ points, encrypts points by CKKS to obtain $[loc]'$ and assigns the identity to these points. Then, $\mathcal{S}_2$ constructs an eRBF tree $\mathbb{EI}'$, which has the same structure with $\mathbb{EI}$. For each node, $\mathcal{S}_2$ randomly generates four vectors $V_1', \ldots, V_4'$ and encrypts them by CKKS. $\mathcal{S}_2$ associates encrypts $[loc]'$ with its corresponding OID or TID in $\mathbb{EI}$. According to secure analysis in [19], a PPT adversary cannot distinguish between the simulated view and the real view.

[Simulating Search] Given $\mathcal{L}_2^{search}(DB_p, Q) = \{Token_i, Token_j, |Q|, S_{can}, S_{ref}, \mathbb{S}(Q), \mathbb{A}(Q), (OID, [loc])_i, (TID, [loc])_j\}$, $\mathcal{S}_2$ randomly generates plain text loc' and encrypts it by using CKKS to get $[loc]'$. From $\mathbb{S}(Q)$, $\mathcal{S}_2$ knows whether a point in query has been searched before or not. From $\mathbb{A}(Q)$, $\mathcal{S}_2$ knows the identifiers of points which are NN points for a query point. If a $q_i \in Q$ is searched before by comparing the token of q_i and in previous tokens, $\mathcal{S}_2$ reuses the previous simulated token and returns the previous NN points as search results. Otherwise, $\mathcal{S}_2$ simulates a new search token $Token'$, which is the token of one point including k hashes $h(kw)$ and a location. Since $\mathcal{S}_2$ knows which leaf node of the eRBF tree matches the search token $Token_j$, $\mathcal{S}_2$ randomly generates a k-bit string as the search token $Token$. The string has the same size as $h(kw)$ and matches with the same leaf node of eRBF. A PPT cannot distinguish between the simulated $Token'$ and the real $Token$. □

5.3. Computational Complexity Analysis

In this section, we analyze the time complexity of RkNNToE, in which the most complexity is caused by computing the distance between two points securely. The complexity of kNN is shown in [19]. To generate the set S_{can}, every query point is checked against nodes and cost $\mathcal{O}(|Q| \cdot (N_{vis}(eRBFtree) + N_{vis}(O_{leaf})))$ at most, where $N_{vis}(eRBFtree)$ is the number of vertexes in the visited nodes and $N_{vis}(O_{leaf})$ is the number of transition points in the leaf nodes that are intersected by the polyline. All filter points are checked against nodes and the cost of computing the distance is $\mathcal{O}(k \cdot |Q| \cdot (N_{vis}(eRBFtree) + N_{vis}(O_{leaf})))$ at most. After obtaining S_{can}, the cost of computing the distances between all transitions in set S_{can} and the query trajectory is $\mathcal{O}(|Q| \cdot |S_{can}|)$. After obtaining S_{ref}, the cost of computing the distances between all transitions in set S_{ref} and their kNN trajectories is $\mathcal{O}(2|S_{\tau'}| \cdot |S_{ref}|)$, where $S_{\tau'}$ is a set of all kNN trajectories of a transition. The total complexity is $\mathcal{O}(RkNNToE) = \mathcal{O}((k+1) \cdot |Q| \cdot (N_{vis}(eRBFtree) + N_{vis}(O_{leaf}))) + \mathcal{O}(|Q| \cdot |S_{can}|) + \mathcal{O}(2|S_{\tau'}| \cdot |S_{ref}|)$. According to [9], the visited nodes are proportional to the number of points in DB_p, f is the fanout of the eRBFtree, and $DB_\tau \ll DB_o$. The complexity is $\mathcal{O}(RkNNToE) = \mathcal{O}((k+1) \cdot |Q| \cdot (N_{vis}(eRBFtree) + N_{vis}(O_{leaf}))) + \mathcal{O}(|Q| \cdot |S_{can}|) + \mathcal{O}(2|S_{\tau'}| \cdot |S_{ref}|) = \mathcal{O}((k+2) \cdot |Q| \cdot (|DB_o|/f))$.

6. Performance Evaluation

In this section, we conduct experiments on the two databases: the aGPS trajectory dataset (Transition dateset) collected in Geolife project in Beijing [22–24] and the bus lines dataset (Trajectory dataset) in Beijing [25]. There are 18,670 transitions in the transition database. The bus lines dataset has 1891 trajectories and 1174 bus stations. All algorithms are implemented in Python language in Windows 10 and examined on a computer with an Intel(R) Core (TM)i5-10505 and 16.00 GB RAM. We randomly generate a query trajectory by selecting an ordered sequence from the trajectory database, since the randomly generated points cannot keep the spatial continuity as a trajectory. In the experiment, the NN k trajectories do not share any one point with the query trajectory. The trajectory that is shared by multiple bus lines is just recorded as one trajectory.

6.1. Constructing eRBF Tree and DList

Before outsourcing the data, the DO needs to build the eRBF tree. The time cost of constructing the eRBF index includes two parts: the time of constructing the RBF tree in database D_τ and the time of constructing the encrypted quad tree in database D_p. The first part is related with the maximum number (N_τ) of trajectory points in a leaf node of the father tree. Table 3 shows the time cost of constructing the RBFs in the father tree with different N_τ. The second part is related to the maximum number (N_o) of transition points in a leaf node of the child tree. The total time of constructing the eRBF tree is shown in Figure 8; the main cost is for encrypting the four vertexes in every node of eRBF tree. With N_τ or N_o increasing, the cost of constructing eRBF decreases, since the DList is constructed based on plain text, and the DO only needs to compute the distance between every transition point with vertexes in its nearby nodes. Here, we set the nodes in the range of 25 to 200 steps, and the mean time of constructing the DList is shown in Table 3.

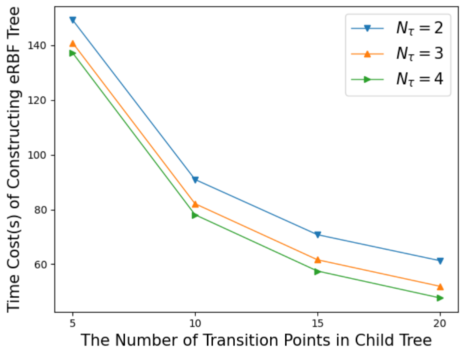

Figure 8. The time cost of constructing the eRBF tree.

Table 3. The cost of constructing father tree and DList.

DB	N_τ	Step Length in Latitude and Longitude	Time Cost of RBF Tree(s)	Time Cost of DList(s)
DB_τ	2	[0.000230, 0.001237]	5.600787	3.559773
	3	[0.000460, 0.002474]	3.796525	3.468612
	4	[0.001840, 0.009896]	2.861059	3.470150
	5	[0.001840, 0.009896]	2.698736	3.444898

6.2. Generating Query

A query of one point includes the encrypted location and an NN search token. The time cost of encrypting a location vector is about 0.004516 seconds by CKKS encryption. The cost of generating a token is related with the search radius. Here, we denote the minimum range of the leaf node in the father tree as a step length and use the number of steps to determine the search radius. The step length does not decrease as N_τ increases, which is shown in Table 3. As shown in Figure 9, the line of "Enc." is the time of encryption of a location. As the number of steps increases, the time of generating a token increases. So, the total time to generate a trajectory query is related to the number of points included in this trajectory and the search radius for every point in the query.

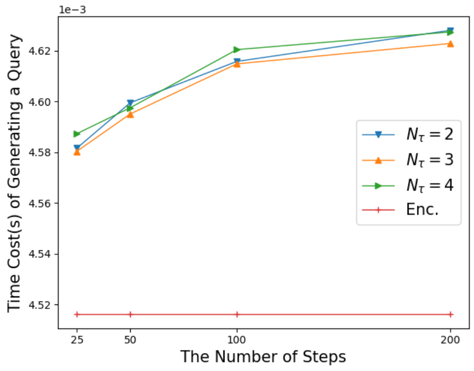

Figure 9. The time cost of generating a query.

6.3. Search

In this section, we firstly demonstrate the time cost of the kNN search for a point. Then we show the total time of two clouds after receiving a RkNNToE request.

6.3.1. NN Trajectories Search

Since the DO needs to search NN trajectories for the refined transitions, it is necessary to illustrate the efficiency of the kNN search for every transition point. As shown in Figure 10, as the number of trajectory points in a leaf node of the father tree increases, the time of searching the NN points increases. As the number of steps in the search radius increases, the time cost of searching NN points increases. The total cost of searching NN trajectories for a transition requires twice as much time as that for NN points in Figure 10.

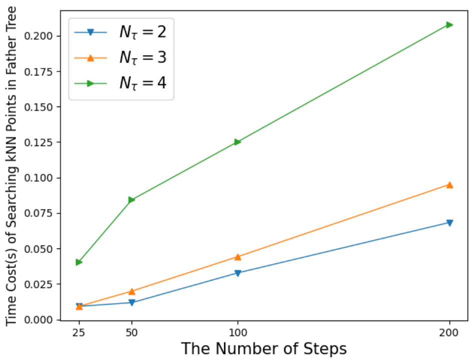

Figure 10. The time cost of searching NN trajectory points.

6.3.2. RkNNToE Search

In this section, we simulate the whole search process in two clouds, which includes finding NN points for every query point, constructing a filter set and pruning transitions, refining transitions and finding NN trajectories for every refined transition. In the simulated search, the eRBF tree is built with $N_\tau = 2$ and $N_o = 2$. The experiment settings are as follows:

The number of points in a query(N_p): 2 to 5, default 3. The k in RkNNToE: 1 to 4, default 2. The number of steps in NN points search: 20 to 200.

The random behavior of a time cost is caused by the random generation query trajectory. The effect of pruning differs widely when the queries are different. According to Section 5.3, the complexity is mainly affected by S_{can} and (DB_O/f) rather than operations of search k NN trajectories. S_{can} and (DB_O/f) are the outputs of pruning, and the size of the filter set does not linearly increase as k increases. In most cases, when $k = 2$, the points in the filter set are a, b, c. $\{a, c\}$ and $\{a, b\}$ can form two trajectories. It also causes the random behavior of time cost. So, we use the median of time cost to analyze the distribution trend of the results. As shown in Figure 11, when the number of points in a query is 3, the median time cost decreases as k increases. As k increases, the number of trajectories in the filter set increases, and the filtered transition increases. In the refining phase, the number of candidate transitions decreases, which leads to the reduction of time. As shown in Figure 12, when $k = 2$, the median of the time cost is increased as the number of points in a query increases. As the number of points in a query increases, the number of points in the trajectories in the filter set increases, which leads to the increased times of computing distance. It also results in the decrease of pruning space, which means the number of refined transitions increases. Both conditions cause the cost time to increase.

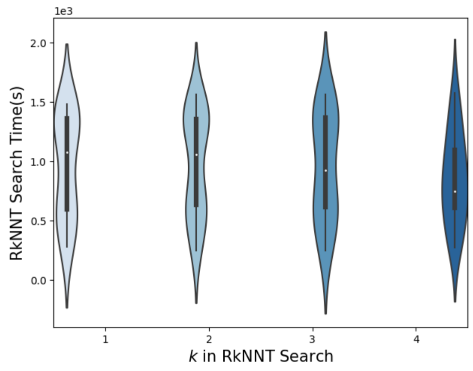

Figure 11. The effect of k in RkNNToE search ($N_P = 3$).

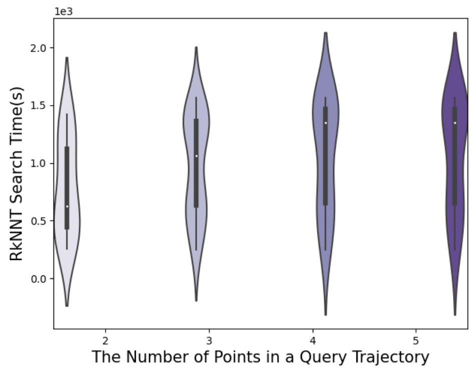

Figure 12. The effect of the number of points in query ($k = 2$).

7. Conclusions

In this paper, we studied a method of route planning on an encrypted trajectory database, RkNNToE, that securely returns all transitions, which are the reverse k nearest neighbor trajectories of the query trajectory. We designed a hybrid encrypted bloom filter tree (eRBFtree) for search in the encrypted trajectory database, which supports space pruning and fast kNN search. Combined with eRBFtree, we gave the pruning strategies to prune the transition as much as possible and to improve the search efficiency. The security analysis showed that the query, data and index are secure in the process of RkNNToE. The experiments showed that RkNNToE can find the results in the RkNNT search efficiently and correctly.

Author Contributions: Conceptualization, X.Z., H.Z. and Q.W.; methodology, X.Z.; software, X.Z.; validation, H.Z., X.Z. and K.L.; formal analysis, X.Z. and H.Z.; investigation, H.Z.; resources, X.Z.; writing—original draft preparation, X.Z.; writing—review and editing, H.Z.; visualization, X.Z. and K.L.; supervision, Q.W. and K.L.; project administration, H.Z.; funding acquisition, H.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (Grant Nos. 62072051, 61976024, 61972048, 62272056).

Data Availability Statement: GPS trajectory dataset (Transition dateset) collected in Geolife project in Beijing [22–24] and the bus lines dataset (Trajectory dataset) in Beijing [25].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, Z.; Shen, H.T.; Zhou, X.; Zheng, Y.; Xie, X. Searching trajectories by locations: An efficiency study. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Indianapolis, IN, USA, 6–10 June 2010; pp. 255–266. [CrossRef]
2. Keogh, E.J. Exact indexing of dynamic time warping. In Proceedings of the VLDB, Hong Kong, China, 20–23 August 2002; pp. 406–417.
3. Vlachos, M.; Gunopulos, D.; Kollios, G. Discovering similar mul- tidimensional trajectories. In Proceedings of the ICDE, San Jose, CA, USA, 26 February–1 March 2002; pp. 673–684.
4. Chen, L.; Özsu, M.T.; Oria, V. Robust and fast similarity search for moving object trajectories. In Proceedings of the SIGMOD, Baltimore, MD, USA, 14–16 June 2005; pp. 491–502.
5. Cheema, M.A.; Zhang, W.; Lin, X.; Zhang, Y.; Li, X. Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. *VLDB J.* **2012**, *21*, 69–95. [CrossRef]
6. Emrich, T.; Kriegel, H.P.; Mamoulis, N.; Niedermayer, J.; Renz, M.; Zufle, A. Reverse-nearest neighbor queries on uncertain moving object trajectories. In *Database Systems for Advanced Applications*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 92–107.
7. Feng, Z.; Zhu, Y. A Survey on Trajectory Data Mining: Techniques and Applications. *IEEE Access* **2017**, *4*, 2056–2067. [CrossRef]
8. Zheng, Y.; Lu, R.; Zhu, H.; Zhang, S.; Guan, Y.; Shao, J.; Wang, F.; Li, H. SetRkNN: Efficient and Privacy-Preserving Set Reverse kNN Query in Cloud. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 888–903. [CrossRef]
9. Wang, S.; Bao, Z.; Culpepper, J.S.; Sellis, T.; Cong, G. Reverse k nearest neighbor serach over trajectories. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 757–771. [CrossRef]
10. Pan, X.; Nie, S.; Hu, H.; Yu, P.S.; Guo, J. Reverse Nearest Neighbor Search in Semantic Trajectories for Location-Based Services. *IEEE Trans. Serv. Comput.* **2022**, *15*, 986–999. [CrossRef]
11. Tzouramanis, T.; Manolopoulos, Y. Secure reverse k-nearest neighbors search over encrypted multi-dimensional databases. In Proceedings of the 22nd International Database Engineering & Applications Symposium (IDEAS), Villa San Giovanni, Italy, 18–20 June 2018; pp. 84–94.
12. Tao, Y.; Papadias, D.; Lian, X. Reverse kNN search in arbitrary dimensionality. In Proceedings of the 30th International Conference Very Large Data Bases, Toronto, ON, Canada, 31 August–3 September 2004; pp. 744–755.
13. Wu, W.; Yang, F.; Chan, C.-Y.; Tan, K.-L. FINCH: Evaluating reverse k-nearest-neighbor queries on location data. *Proc. Vldb Endow.* **2008**, *1*, 1056–1067. [CrossRef]
14. Lu, J.; Lu, Y.; Cong, G. Reverse spatial and textual k nearest neighbor search. In Proceedings of the ACM SIGMOD International Conference on Management Data, Athens, Greece, 12–16 June 2011; pp. 349–360.
15. Lu, Y.; Cong, G.; Lu, J.; Shahabi, C. Efficient algorithms for answering reverse spatialkeword nearest neighbor queries. In Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, Bellevue, WA, USA, 3–6 November 2015; pp. 1–4.
16. Pournajaf, L.; Tahmasebian, F.; Xiong, L.; Sunderam, V.; Shahabi, C. Privacy preserving reverse k-nearest neighbor queries. In Proceedings of the 19th IEEE International Conference Mobile Data Manage, (MDM), Aalborg, Denmark, 25–28 June 2018; pp. 177–186.
17. Li, X.; Xiang, T.; Guo, S.; Li, H.; Mu, Y. Privacy-preserving reverse nearest neighbor query over encrypted spatial data. *IEEE Trans. Serv. Comput.* **2022**, *15*, 2954–2968. [CrossRef]
18. Wang, Q.; He, M.; Du, M.; Chow, S.S.; Lai, R.W.; Zou, Q. Searchable encryption over feature-rich data. *IEEE Trans. Dependable Secur. Comput.* **2016**, *15*, 496–510. [CrossRef]
19. Lei, X.; Tu, G.H.; Xie, A.X.L.T. Fast and Secure kNN Query Processing in Cloud Computing. In Proceedings of the 2020 IEEE Conference on Communications and Network Security (CNS), Avignon, France, 29 June–1 July 2020.
20. Finkel, R.A.; Bentley, J.L. Quad trees a data structure for retrieval on composite keys. *Acta Inform.* **1974**, *4*, 1–9. [CrossRef]
21. Lindell, Y. How to simulate it—A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 277–346.
22. Zheng, Y.; Zhang, L.; Xie, X.; Ma, W. Mining interesting locations and travel sequences from GPS trajectories. In Proceedings of the International conference on World Wild Web (WWW 2009), Madrid, Spain, 20–24 April 2009; ACM Press: New York, NY, USA, 2009; pp. 791–800.
23. Zheng, Y.; Li, Q.; Chen, Y.; Xie, X.; Ma, W. Understanding Mobility Based on GPS Data. In Proceedings of the ACM conference on Ubiquitous Computing (UbiComp 2008), Seoul, Republic of Korea, 21–24 September 2008; ACM Press: New York, NY, USA, 2008; pp. 312–321.

24. Zheng, Y.; Xie, X.; Ma, W. GeoLife: A Collaborative Social Networking Service among User, location and trajectory. *IEEE Data Eng. Bull.* **2010**, *33*, 32–40.
25. BeiJIngBusStation. Available online: https://github.com/FFGF/BeiJIngBusStation (accessed on 7 March 2023).