Modeling and Querying Similar Trajectory in Inconsistent Spatial Data

Weijia Feng[1,2](✉), Yuran Geng[1], Ran Li[1], Maoyu Jin[1], and Qiyi Tan[1]

[1] Tianjin Normal University, Binshui Xi Road 393, XiQing District, Tianjin, China
WeijiaFeng@tjnu.edu.cn

[2] Postdoctoral Innovation Practice Base Huafa Industrial Share Co., Ltd., Changsheng Road 155, ZhuHai, China

Abstract. Querying clean spatial data is well-studied in database area. However, methods for querying clean data often fail to process the queries on inconsistent spatial data. We develop a framework for querying similar trajectories inconsistent spatial data. For any given entity, our method will provide a way to query its similar trajectories in the inconsistent spatial data. We propose a dynamic programming algorithm and a threshold filter for probabilistic mass function. The algorithm with the filter reduces the expensive cost of processing query by directly using the existing similar trajectory query algorithm designed for clean data. The effectiveness and efficiency of our algorithm are verified by experiments.

Keywords: Similar trajectory query · Spatial data · Database

1 Introduction

In real world, spatial data is often inconsistent, such as data collected from location-based services [1, 2], data integration [3, 12], and objects monitoring [4]. Methods for querying such kind of spatial data are not yet well developed. Query processing in inconsistent data is extremely expensive by a straightforward extension of existing methods for clean data [6, 9]. To develop efficient query processing methods for inconsistent spatial data, we in this paper study frequent nearest neighbor query which is a very time costing problem even in the context of clean data.

1.1 Modeling Inconsistent Spatial Data

Inconsistent spatial data overruns everywhere. For example, due to the inaccuracy of measurements, it is hard to obtain the concrete location of an entity [5, 6, 8]. To deal with this, the spatial information of an entity is modeled as a distribution over some local area. Besides, data is often integrated from different sources, thus causing the hardness to obtain the right distribution of an entity. Therefore, in real applications, a spatial entity is usually represented by a finite set of potential specific spatial distributions. For example, a real-life entity E is composed of N inconsistent distributions, that is, $E = \{e_1, \cdots, e_N\}$, where e_i represents the distribution from the i-th data source.

In this paper, we use this popular model, *i.e.*, probabilistic data, to model the inconsistent data and then study the query algorithm on it. We associate e_i area with the data collected in t time probability $reg^t(e_i)$ model. In this paper, we use a collection of multiple discrete sampling locations where s_i^t is the entity data collected in t time. Every sampling location $S_{i,j} \in s_i^t$ has a probability of its occurrence which can be written as $\mathbf{Pr}(S_{i,j}) \in (0, 1]$. The occurrence probability $\mathbf{Pr}(S_{i,j})$ should satisfy the requirement $\sum_{j=1}^{m} \mathbf{Pr}(S_{i,j}) = 1$. The sets of entity collection data in E constitute the probability database D^p, and the data set collected by the entity at a given time T is called the sketch of an entity at time T.

In the real application, probability data often associates with regional correlation. For example, in the spatial data, the physical quantities such as temperature and light measured from the nearby locations are probabilistic, and these data are also very similar, which is the regional correlation of the data. Regional association probability entity data contains several non-overlapping association regions. The entity $e_i \in E$ belongs to only one Inconsistent Regional Circle (*IRC*), and the entity set in the same *IRC* as e_i is denoted as the regional association set $IRC(e_i)$, where $IRC(e_i)$ contains e_i. We assume that entities in the same *IRC* are independent of each other, and entities in different *IRC* are independent of each other, and therefore, the data acquired from practices are independent of each other.

We use Bayesian network to express the regional correlation of data. Bayesian networks are able to represent the dependencies between entities as acyclic directed graphs, and each entity has a conditional probability table, which represents the conditional probability under the joint distribution with the parent of the entity.

In probabilistic data model, the probabilistic entity e_i uncertainly becomes the nearest neighbor of a query position q. The probability $\mathbf{Pr}_{sim\text{-}point}$ (e_i, q) is used to represent the probability that e_i becomes the nearest neighbor of the query location q. Then the probability nearest neighbor query *sim-point* on the regional correlation probability data is to find the entity whose $\mathbf{Pr}_{sim\text{-}point}$ exceeds the given threshold constraint. A trivial method is to access the conditional probability table of e_i, and using the variable elimination method to obtain the joint probability, and then decide if e_i belongs to the query result.

1.2 Nearest Entities in Inconsistent Spatial Data

In the real world, spatial data is usually changing. For example, data obtained from sensor nodes for the environment monitoring keeps changing over time. At different time, the data collected by sensor nodes is different. Results of a *sim-traj* query in sensor network is the entities with high probability frequently appearing in the result of probabilistic nearest neighbor query in multiple sketches represented at different moments.

This problem can be described as follows. Given the positive integer $k > 0$ and the threshold $\delta > 0$, then input t sketches $D_1^p, \cdots, D_t^p$ with the query positions $q_1, \cdots q_t$. The *sim-traj* query outputs a collection of entities such that for each entity in the collection, the probability of being the nearest neighbor to the query location greater than or equal to k is greater than the threshold δ.

Let N^{e_i} be the number of sketches where e_i becomes the nearest neighbor of the query locations. For example, $N^{e_1} = 3$ implies the event "e_i is becomes the nearest neighbor of the query location three times".

We suppose that $\mathbf{Pr}(N^{e_1} \geq k) = \sum_{i=k}^{\infty} \mathbf{Pr}(N^{e_1} = i)$ which is the probability mass function of N^{e_1}. The result of a *sim-traj* query is a collection of entities that satisfies $\mathbf{Pr}(N^{e_1} \geq k) \geq \delta$. A *sim-point* query is a query carried out on the sketch D_t^P corresponding to a moment t in the probabilistic data set D^P. Therefore, a *sim-point* query can be resolved as follows: let be $\alpha = 0$ be the threshold of the *sim-traj* query, then evaluate the query over each sketch by *find-sim-traj* for entities e_j in any sketch D_i^P to obtain the probability p_{ji} of becoming the nearest neighbor of query position q_i. Then, According to the result of t and 2^t different combinations of calculate the probability distribution of random variables N^{e_i}, and thus decide if entity e_i is in the query result. However, such a trivial method may lead to a large time overhead, reasons can be listed as follows,

1. Let the value of *sim-traj* threshold be $\alpha = 0$, *find-sim-traj* may degenerate to trivial search on indexes due to the failure of the upper bound pruning method.
2. Accessing conditional probability tables results in a lot of time overhead, as we can see from the experiments on the pruning method proposed in this paper, a lot of access is unnecessary.
3. Each entity e_i requires an exponential time overhead to compute the probability distribution of N^{e_i}.

As discussed above, the existing work cannot effectively handle *sim-traj* queries. The inadequacy of the existing work implies the significance of our method for the *sim-traj* query processing.

1.3 Contributions

In this paper, we study *sim-traj* query problem. We use probabilistic data to model inconsistent spatial data. A kind of frequent probabilistic nearest neighbor query is studied in-depth in this paper. We study the probabilistic data of regional association by using multiple sketches and propose a framework to find frequent probability nearest neighbor. A dynamic programming algorithm is designed to compute the square time of the probabilistic mass function. Based on the DP algorithm for *sim-traj* queries, a basic processing algorithm is proposed; And then we develop an efficient pruning methods which are used to reduce the search space of *sim-traj* queries. Finally, a series of experiments are conducted on both the synthetic data set and the real data set, and experimental results show that the efficiency of our algorithm.

2 Problem Definition

Definition 1 (*similar-point-probability*). Let $E = \{e_i\}_{i=1}^{N}$ be a collection of probabilistic entities, and $D^P = \{s_i\}_{i=1}^{N}$ the sample data of entity set E at some time. Given position q, let r_1 and r_2 be the possible nearest and farthest distance between probabilistic entity e_i and q (depends on the sampling situation of the corresponding region $reg(e_i)$) and

L_2-norm $d(\cdot, \cdot)$. Then $\mathbf{Pr}_{sim\text{-}point}\ (q, e_i)$ is the probability that entity e_i becomes the nearest neighbor of query location q as

$$\mathbf{Pr}_{sim-point}(q, e_i) = \int_{r_2}^{r_1} \mathbf{Pr}(d(q, e_i) = r) \times \mathbf{Pr}\Big(\wedge_{\forall u \in IRC(e_i)\backslash\{\mathrm{e_i}\}} d(q, u) \geq r | r\Big) \times \mathbf{Pr}\Big(\wedge_{\forall v \in E \backslash \mathrm{IRC}(\mathrm{e_i})} d(q, v) \geq r\Big)\mathbf{d}r$$

Definition 2 (*similar-point*). Let $E = \{e_i\}_{i=1}^N$ be a collection of probabilistic entities, and $D^P = \{s_i\}_{i=1}^N$ the sample data of entity set E at some time. Given threshold α and query location q, the result of a *sim-point* query (q, α) is a set of probabilistic entities such that $\gamma = \big\{e_i : \mathbf{Pr}_{sim-point}(q, e_i) > \alpha,\ i \in [1, N]\big\}$.

In this paper, the random indicator variable $I_{ij} = 1$ is used to indicate that e_j becomes *sim-point* of q_i. At the same time, we denote $\mathbf{Pr}\big(I_{ij} = 1\big)$ as $\mathbf{Pr}\big(I_{ij}\big)$ for short, hence, $\mathbf{Pr}\big(I_{ij}\big) = \mathbf{Pr}_{sim-point}(q, e_i)$. Now, we formally define the frequent probabilistic nearest neighbor query on regional associated probabilistic data.

Sim-traj query. Let $E = \{e_i\}_{i=1}^N$ be a collection of probabilistic entities, and $D^P = \{s_i\}_{i=1}^N$ the sample data of entity set E at some time. Given a positive integer k, the threshold value δ, and the set containing t query locations $Q = \{q_i\}_{i=1}^t$, the result of a *sim-traj* query Q is $\gamma = \big\{e_i : \mathbf{Pr}_{sim-point}(N^{e_i} \geq k) \geq \delta,\ i \in [1, N]\big\}$.

Obviously, when the parameters $t = 1$, $k = 1$, $\alpha = \delta$, any *sim-traj* query is equivalent to a *sim-point* query. Therefore, the problem studied in this paper is actually a generalized version of *sim-point* query.

3 Frequent Probabilistic Nearest Neighbor Query Processing

We begin with an overview of query processing that deals with frequent probability nearest neighbors,

1. For the given t sketches $D_1^p, \cdots, D_t^p$ to establish corresponding R-star index tree $index_1, \cdots, index_t$. In detail, and in all probability will all *IRC* entities into the corresponding minimum circumscribed rectangle, insert each *IRC*, so as to establish indexes;
2. For the current set of input query location, the *find-sim-point* method proposed in literature [7] was used, by setting α = 0. Traverse the indexes and obtaining the upper bound of *sim-point* probability of all entities relative to t query locations. Use the first pruning condition given below in the next subsection to filter out the invalid entities.
3. Access and compute the conditional probability table for the remaining entities, to obtain the *sim-point* probability relative to t query locations
4. At last, for every candidate entity left e_j, compute the exact probability $\mathbf{Pr}(N^{e_j} \geq k)$ that it becomes a *sim-traj*, and return the query results.

Next, we detail how to calculate the probability $\mathbf{Pr}(N^{e_j} \geq k)$, in other words, we give a method to calculate the joint nearest neighbor probability efficiently under the condition that the *sim-point* probability $\mathbf{Pr}\big(I_{ij}\big)$ of e_j is known in advance.

3.1 The Computation of the Nearest Neighbor Probability

As mentioned earlier, a trivial algorithm leads to an exponential time cost. In this section, we design the dynamic programming algorithm which can derive the probability in quadratic time while avoiding the computational cost.

First, the cumulative probability $\mathbf{Pr}_{\geq s,l}(e_j)$ is defined for each probabilistic entity e_j, its meaning is that there are more than s positions in the given l positions (respectively $q_1 \cdots, q_l$), then, the sim-point probability of e_j can be formulated as

$$\mathbf{Pr}_{\geq s,l}(e_j) = \sum_{Q' \subseteq Q_l, |Q'| \geq s} \left(\prod\nolimits_{q_i \in Q'} \mathbf{Pr}(I_{ji}) \times \prod\nolimits_{q_i \in Q_l \setminus Q'} \left(1 - \mathbf{Pr}(I_{ji})\right) \right)$$

Then, let $\mathbf{Pr}_{\geq 0,l}(e_j) = 1, \forall 0 \leq l \leq t$, and $\mathbf{Pr}_{\geq s,l}(e_j) = 1, \forall 0 \leq l \leq s$.

Time Complexity. For each candidate probability entity e_j, the dynamic programming algorithm costs $O(t^2)$ space cost and $O(t^2)$ time to calculate $\mathbf{Pr}(N^{e_j} \geq k)$.

3.2 Frequent Probability Nearest Neighbor Lookup Basic Query Algorithm

The literature [7] proposed *sim-point* query processing algorithm *find-sim-point*. Given the probabilistic entity database D^P, R-star index tree, location query q and threshold α, it takes the best first search strategy to traverse the index. During the traversal, the algorithm pruned the invalid entities by using the upper and lower bounds of the probability recorded in the index, and finally calculated the exact value of $\mathbf{Pr}_{\textit{sim-point}}$ for the remaining candidate entities successively. The aim of this paper is to calculate the exact value of $\mathbf{Pr}_{\textit{sim-point}}$ for all entities. For this purpose, let the threshold $\alpha = 0$. This algorithm can get the accurate value $\mathbf{Pr}_{\textit{sim-point}}$ for each entity. After all the probability values are obtained by calling the above algorithm, the dynamic programming algorithm introduced in the previous section is used to calculate the $\mathbf{Pr}_{\textit{sim-point}}$. We present a basic algorithm **Baseline** to compute a *sim-traj* query as shown in Fig. 1.

Time Complexity of Baseline Algorithm. The dynamic programming algorithm in steps 5 to 10 has a time cost of $O(Nt^2)$. The time overhead of steps 1 to 3 comes from the computation of N_t probability values $\mathbf{Pr}(I_{ij})$. This needs to access the conditional probability table in the regional association probability database which is very expensive. Therefore, we next propose the pruning methods.

3.3 Probability Nearest Neighbor Advanced Query Algorithm

The main time overhead of the Baseline algorithm is related to the number of entities to be precisely calculated for the probability. Therefore, the next section focuses on pruning in steps 2 and 4 of query processing. In order to reduce substantial time, the number of entities to be precisely calculated is reduced at a lower cost as possible. Next, we explain the pruning conditions in steps 2 and 4, which are called first pruning and second pruning, in order to improve the basic algorithm mentioned above.

```
Input:   entity set $E = \{e_i\}_{i=1}^{N}$, sketch set $\{D_i^P\}_{i=1}^{t}$, R star tree index $\{\text{index}_i\}_{i=1}^{t}$,
         query location set $\{q_i\}_{i=1}^{t}$, positive integer $k$, threshold $\delta \in (0,1]$
Output: sim-traj query result collection $\gamma$
1 foreach $i \le t$ do
2     Call subprocess find-sim-point$(\text{index}_i, q_i, 0)$ to get all $p(I_{ji})$
3 end
4 foreach $e_j \in E$ do
5     Calculate $\mathbf{Pr}_{sim\text{-}traj}(e_j)$ (dynamic programming algorithm)
6     if $p_{sim\text{-}traj}(e_j) \ge \delta$ then
7       $\gamma = \gamma \cup \{e_j\}$
8     end
9 end
10 return $\gamma$
```

Fig. 1. Baseline algorithm

To calculate the exact value of each $\mathbf{Pr}(I_{ij})$. The algorithm in step 2 must consume a lot of access time to access and calculate the relevant conditional probability table. Therefore, it is necessary to propose efficient pruning methods to improve performance.

As long as the pruning condition filters most of the candidate entities, it can significantly reduce the number of entities that need to be accurately calculated for the probability $\mathbf{Pr}(I_{ji})$. Every time a candidate entity is pruned, the time saved is at least t times the calculation time of the exact value. We can see from the follow-up experiments in this paper, the pruning effect of this method is very obvious, which greatly improves the query efficiency. Step 2 in the index phase, we can prune a large number of probability entities to calculate the exact value of *sim-point* by using the upper bound of *sim-point* derived below. When the upper bound of probability $\mathbf{Pr}(I_{ji})$ can be calculated with less online overhead, the pruning method can be used, hence, the baseline algorithm is improved. An index pruning criterion based on Chernoff bounds can be expressed as follows.

Theorem 1. *For the probability entity* e_j, *given* $k(\le t)$, *threshold* $\delta \in (0,1]$ *and upper bound set* $\left\{\widehat{\mathbf{Pr}}_{ji}\right\}_{i=1}^{t}$. *If* $\sum_{i=1}^{t}\widehat{\mathbf{Pr}}_{ji} < k$ *and* $\sum_{i=1}^{t}\widehat{\mathbf{Pr}}_{ji} \ge k\left(\ln\left(\sum_{i=1}^{t}\widehat{\mathbf{Pr}}_{ji}\right) - \ln k + 1\right) - \ln \delta$, *then* $e_j \notin \gamma$.

Each pruning of an entity saves the time cost of accurate calculation of I_{ji} and verification of $\mathbf{Pr}(N^{e_j}) \ge k$.Therefore, by the above Theorem 1, the query processing algorithm is improved as follows: (1) For each entity e_j and the corresponding query location q_i, the upper bound $\widehat{\mathbf{Pr}}_{ji}$ is quickly calculated when traversing the index. (2) Using the formula given in Theorem 1 to prune the invalid entity.

The final experiment in this paper verifies that the actual performance of pruning Theorem 1 is efficient.

Algorithm 3–2 gives the frequent probability neighbor query processing algorithm *find-sim-traj*.

1. Line 1 to 3 initialize each probability upper bound as 0.
2. Lines 4 to 9 traverse the index of each sketch by calling the *find-sim-point* method. In this way, the upper bound of probability of each entity can be updated in the process of traversing, and the corresponding upper bound of probability of each entity can be obtained after traversing t indexes.
3. Line 10 uses Theorem 1 for the pruning.
4. From line 11 to 13, the accurate probability of the remaining entities is calculated by accessing the sketch data.
5. Lines 14 to 19 run the dynamic programming algorithm, compute the corresponding probability of entities that are not pruned during the second time, and output the query result γ.

```
Input:  entity set $E = \{e_i\}_{i=1}^{N}$, sketch set $\{D_i^P\}_{i=1}^{t}$, R star tree index $\{\text{index}_i\}_{i=1}^{t}$,
        query location set $\{q_i\}_{i=1}^{t}$, positive integer $k$, threshold $\delta \in (0,1]$
Output: *sim-traj* query result collection $\gamma$
1  foreach $e_j \in E$ do
2      Initialization $\widehat{\mathbf{Pr}}_{total}(e_j) \leftarrow 0$
3  end
4  foreach $i \le t$ do
5      $\beta$=*find-sim-point*$(\text{index}_{i,}\ q_{i,}\ 0)$
6      foreach $e_j \in \beta$ do
7          $\hat{\mathbf{Pr}}_{\text{total}}(e_j) = \hat{\mathbf{Pr}}_{\text{total}}(e_j) + \hat{\mathbf{Pr}}_{ji}$
8      end
9  end
10 Perform the first pruning by Theorem 1 and the remaining $\beta$
11 foreach $e_j$   $\beta$ do
12     Calculate the value of $\mathbf{Pr}(I_{ji})$
13 end
14 foreach $e_j \in \beta$ do
15     if $\mathbf{Pr}_{sim\text{-}traj}(e_j) \ge \delta$ then
16         $\gamma = \gamma \cup \{e_j\}$
17     end
18 end
19 return $\gamma$
```

Algo. 3-2 *sim-traj* query processing algorithm

3.4 The Calculation of the Probability Upper Bounds

As mentioned before, if the above theorem can be applied in practice, we also need to know how to get the upper bound of *sim-traj* probability for each entity to become a query location by fast online calculation. Therefore, two off-line precomputation methods are presented in the next section. By embedding the structure in the index to save the online time of calculating the upper bound, the above pruning method is used to make the query processing algorithm run in practice.

The upper bound of the exact value of probability $\mathbf{Pr}(I_{ij})$ of entity e_j in quick sketch D_i^P is

$$\widehat{\mathbf{Pr}}_{ji} = \sum\nolimits_{\forall s_{jl} \in s_j^i \wedge \lambda = max\left\{\lambda' | \lambda' \le d(\mathrm{q_i}, \mathrm{s_{jl}}) - d\left(q_i, piv_{s_{jl}}\right)\right\}} \mathbf{Pr}_{\mathrm{J}}(s_{jl}, \lambda),$$

where the joint probability is

$$\mathbf{Pr}_{\mathrm{J}}(s_{jl}, \lambda) = \mathbf{Pr}\left\{\wedge_{\forall u \in IRC(e_j)} d\left(piv_{s_{jl}}, u\right) \ge d(q_i, s_{jl}) - d\left(q_i, piv_{s_{jl}}\right)\right\}$$

Here, if the query location q_i is not given, then the exact value of the inequality variable cannot be obtained. However, we find that the endpoints of the numerical range can be pre-estimated. That is to say, for each sampling position s_{jl} of each entity e_j, first select a value λ in the interval $[\lambda_{\min}, \lambda_{\max}]$, and at the same time, select a pivot $piv_{s_{jl}}$, then $\mathbf{Pr}_{\mathrm{J}}(s_{jl}, \lambda)$ can be calculated offline in advance. When the query location q_i arrives, the algorithm only calculates the value of $b = d(q_i, s_{jl}) - d\left(q_i, piv_{s_{jl}}\right)$ temporarily, then it can find the required $\mathbf{Pr}_{\mathrm{J}}\left(s_{jl}, \max_{\lambda}\{\lambda < b\}\right)$.The sum of such upper bound probability values corresponding to each sampling position is the upper probability bound $\hat{p}_{ji}$ of Theorem 1.

In order to select the pivot, the data space is divided into rectangles with side length $\varepsilon \in (0, \varepsilon_{\max}]$.Each sampling position s_{jl} selects four rectangular endpoints as alternative pivots for offline prediction. When the query position q_i comes, the algorithm can select the candidate points which are in the same quadrant as q_i relative to s_{jl} as pivots, and calculate the joint probability $\mathbf{Pr}_{\mathrm{J}}(s_{jl}, \lambda)$, so as to obtain the upper bound value. Specifically, the algorithm usually selects c preset ε value $\varepsilon_1, \ldots, \varepsilon_c$ as an alternative parameter in a small interval $(0, \varepsilon_{max}]$ uniformly, randomly and without playback in real data. Then, corresponding to each position, the optional pivots are calculated offline under each optional parameter, and the number is 4 c for each optional parameter in turn. Once the query position q_i is given, the algorithm first finds the pivots in the same block as the query position in the different alternative parameters ε and then temporarily computes their corresponding upper bounds, which results in c alternative upper bounds.

4 Experiments

In this paper, extensive experiments are conducted on real and synthetic regional association probability datasets to investigate the proposed query processing method.

4.1 Experimental Configuration

All experimental configurations are the same as in the previous section. The generation method given in the literature [11] is used here to generate the artificial data set. The distribution of a given number of query positions has a great impact on the results of a *sim-traj* query Querying randomly on t randomly *sim-traj* generated sketches is usually not practical. Thus, in order to facilitate us to comprehensively investigate the performance of the algorithm, we experimentally query *sim-traj* on t replicas of the dataset D^p. Again, we examine the query on t queries with a close distribution of positions. Thus, in the experiment, we generate randomly t points with a high concentration of positions in a relatively small interval r. Let δ be 0.1/0.2/0.4/0.8, k be 5/10/20, t be 20, r be 0.01, N be 10, and d be 5.

To generate the sketch set D^p, 200, 000 points are randomly selected in the interval $R = [0, 10]^d$ and these points correspond to the reference points of *IRC*. On this basis, we construct the dimension of the entity, randomly select the real number a from the interval [1, 4], and set a to the region size of the *IRC*. We further randomly generate n probabilistic entities in the region of *IRC* according to two distributions, where the probability distributions we choose are the most common random probability distribution and the Gaussian probability distribution. Finally, we construct an acyclic directed graph of the *IRC* on the *IRC* and generate a joint probability distribution of the entities. We examine the performance of the algorithm on the U.S. traffic data whose total number of NR is 20,000 [11]. The experiment takes the geographic coordinates as the center point of the *IRC* and expands it as above into a set of available probabilistic data. Based on different distribution functions, the generated datasets can be classified into uniform synthetic data (UniS), traffic data (UniT) and Gaussian traffic data (GauT).

4.2 Analysis of Experimental Results

The comparison algorithm chosen for the experiments in this paper is the **Baseline** Algorithm and its corresponding improvement algorithm. The experiments focus on the efficiency and speed-up ratio of the algorithm for different environmental parameters. Here, we define the speed-up ratio $\eta = \frac{t_{Baseline}}{t_{sim-traj}}$, where $t_{Baseline}$ represents the runtime of the **Baseline** algorithm and $t_{sim\text{-}traj}$ represents the runtime of the *sim-traj* query processing algorithm.

Here, we examine the pruning effect of the pruning strategy on different datasets. Fig. 2 clearly shows that the pruning strategy proposed in this paper can greatly improve the efficiency of the algorithm and reduce the computation time. When pruning a candidate set for the first time using the upper boundary condition, the wrong entities selected by the **Baseline** algorithm can be pruned out, avoiding unnecessary time spent on subsequent calculations of joint probability distributions and boundary probabilities.

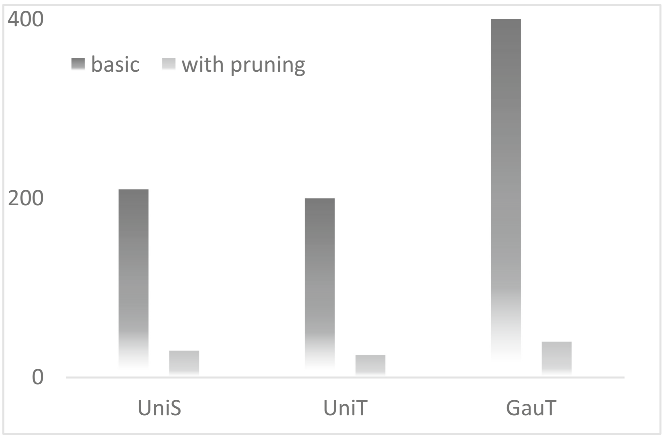

Fig. 2. Effectiveness of pruning criterions

5 Conclusion

In this paper, a general processing framework is proposed including dynamic programming algorithm and threshold filtering method for probabilistic mass function which solves the problem that it is expensive to deal with the query directly by using the existing nearest neighbor query algorithm based on traditional clean data. The effectiveness and efficiency of the algorithm are verified by experiments . Therefore, the work of this paper overcomes the shortcoming that the existing methods are too strict to query results, as many query results as possible are given with custom quality assurance are be ensured.

Acknowledgement. This research is funded by the Natural Science Foundation of China (NSFC): 61602345; National Key Research and Development Plan: 2019YFB2101900; Application Foundation and Advanced Technology Research Project of Tianjin (15JCQNJC01400).

References

1. Chen, L., Thombre, S., Järvinen, K., et al.: Robustness, security and privacy in location-based services for future IoT: a survey. IEEE Access **5**, 8956–8977 (2017)
2. Zheng, X., Cai, Z., Li, J., et al.: Location-privacy-aware review publication mechanism for local business service systems. In: IEEE INFOCOM 2017-IEEE Conference on Computer Communications, pp. 1–9. IEEE (2017)
3. Stonebraker, M., Ilyas, I.F.: Data integration: the current status and the way forward. IEEE Data Eng. Bull. **41**(2), 3–9 (2018)
4. Cheng, S., Cai, Z., Li, J.: Approximate sensory data collection: a survey. Sensors **17**(3), 564 (2017)
5. Miao, D., Cai, Z., Li, J., et al.: The computation of optimal subset repairs. Proc. VLDB Endowment **13**(12), 2061–2074 (2020)

6. Bertossi, L.: Database repairs and consistent query answering: origins and further developments. In: Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, pp. 48–58 (2019)
7. Lian, X., Chen, L., Song, S.: Consistent query answers in inconsistent probabilistic databases. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 303–314 (2010)
8. Wijsen, J.: Foundations of query answering on inconsistent databases. ACM SIGMOD Rec. **48**(3), 6–16 (2019)
9. Greco, S., Molinaro, C., Trubitsyna, I.: Computing approximate query answers over inconsistent knowledge bases. IJCAI **2018**, 1838–1846 (2018)
10. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis. Cambridge university press (2017)
11. Chen, L., Lian, X.: Query processing over uncertain and probabilistic databases. In: Lee, S.-G., Peng, Z., Zhou, X., Moon, Y.-S., Unland, R., Yoo, J. (eds.) DASFAA 2012. LNCS, vol. 7239, pp. 326–327. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29035-0_32
12. Miao, D., Liu, X., Li, J.: On the complexity of sampling query feedback restricted database repair of functional dependency violations. Theoret. Comput. Sci. **609**, 594–605 (2016)