Top-k Distance-Based Outlier Detection on Uncertain Data

Ying Zhang, Hongyuan Zheng(✉), and Qiulin Ding

College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
984589338@qq.com, zixiayedu@126.com, qlding@nuaa.edu.cn

Abstract. In recent years, more researchers are studying uncertain data with the development of Internet of Things. The technique of outlier detection is one of the significant branches of emerging uncertain database. In existing algorithms, parameters are difficult to set, and expansibility is poor when used in large data sets. Aimed at these shortcomings, a top-k distance-based outlier detection algorithm on uncertain data is proposed. This algorithm applies dynamic programming theory to calculate outlier possibility and greatly improves the efficiency. Furthermore, an efficient virtual grid-based optimization approach is also proposed to greatly improve our algorithm's efficiency. The theoretical analysis and experimental results fully prove that the algorithm is feasible and efficient.

Keywords: Uncertain data · Outlier detection · Dynamic programming theory · Optimization approach

1 Introduction

In the real world, data contains uncertainty for reasons that include poor precision of equipment, absence of data and transmission delay. The new data pattern has become a new research hotspot, outliers detection is one of them. Outlier detection is a fundamental issue in data mining. It has been applied in lots of areas including network intrusion detection [1], industrial sciences [2], environmental monitoring [3], credit card fraud detection [4], etc. The outlier is data with abnormal behavior or characteristic that obviously deviates from other sample data. The main task of outlier detection is to find outlier from a large number of complex dataset [5].

The outlier detection comes from the statistics, and later it was introduced into the data mining domain by Knorr [6, 7]. Some outlier detection algorithms are commonly used at present, such as distance-based outlier detection and density-based outlier detection. However, all these outlier detection algorithms mainly focus on deterministic data and cannot directly process uncertain data, most existing outlier detection algorithms on uncertain data are the improvement and innovation on the basis of the above algorithms.

At present, the outlier detection technology of deterministic data has matured and is used cosmically, but research about uncertain data is just beginning. The first definition of outlier on uncertain data was given by Aggarwal C C, Yu P S, et al. [8]. According to them, an uncertain data that can be represented by a probability density function (PDF) is an outlier if its existential probability in some subspace with density at least η is less than δ. But their works only can detect local outlier and cannot be well applied to global outlier. Shaikh and S.A also proposed an outlier detection algorithm on uncertain data of the Gaussian distribution in [9]. According to them, for an uncertain data, if its expected number of neighbors that exist within its D-distance is less than or equal to threshold, it is a density-based outlier. However, in practice, parameter p is hard to set. Subsequently, aimed at the shortcomings of the proposed algorithms in [9], they proposed a fast top-k outlier detection algorithm on uncertain data in [10] and an approximate outlier detection algorithm in [11]. However, all of their works are based on an uncertain data model that Gaussian probability density function is used to describe uncertain data, which is different from the data model we will discuss in this paper. Wang et al. first proposed the concept of distance-based outlier detection. In [12], the definition of outliers is based on the possible world. They judge whether an object is an outlier by calculating the sum of the probabilities of possible world instances that consist of at least k objects. If the sum is less than a pre-defined value, the object is an outlier, which is then extended to outlier detection on the uncertain data stream in [13].

Existing distance-based outlier detection algorithms need user to set up several parameters in practical applications, but it is very difficult to set up reasonable parameters for users who are not familiar with data distribution. In order to decrease the difficulty of parameter setting, we use a constant parameter k that is independent of the dataset features to instead of probability threshold value. In this paper, we redefine the concept of outlier on uncertain data and propose a top-k distance-based outlier detection algorithm on uncertain data. We sort all objects in descending order of its probability of being outlier and the preceding k objects are our results. In order to reduce the computational cost, we apply dynamic programming theory to compute the possibility of being an outlier for an object and greatly improve the efficiency, then present an efficient virtual grid-based optimization method to achieve better effect.

The remainder of this paper is organized as follows. In the next section, we propose a basic algorithm of top-k distance-based outlier detection on uncertain data. A grid-based optimization approach was presented to improve the efficiency of the basic algorithm in Sect. 3. Section 4 contains experimental results that fully prove the efficiency of the proposed algorithms and Sect. 5 concludes our paper.

2 Top-K Distance-Based Outlier Detection on Uncertain Data

2.1 Related Concepts

In this paper, our works focus on tuple-level uncertain data. Given an uncertain dataset $D = \{x_0, x_1, \ldots, x_i, \ldots\}$, in which each tuple x_i has the form of $<\overline{xi}, pi>$, where $\overline{xi}$ is uncertain record of x_i, p_i is existential probability of x_i. $|D| = N$.

Definition 1 (ud-neighborhood [12]). *ud* is a distance parameter. An object is a neighbor of x_i if the distance between it and x_i is less than *ud*. $R(x_i)$ is ud-neighborhood of x_i, $R(xi) = \{xj|xj \in D, dist(\overline{xi}, \overline{xj}) \leq ud\}$.

The main difference between certain and uncertain data is that uncertain data contain uncertain record and their existential probability. So any combination of tuples in ud-neighborhood of the data needs to be unfolded. Then possible world model was built. There are many models of uncertain data, but possible world model is the most popular one [14].

Definition 2 (Probability of the possible world instance (*PPW*)). For uncertain data x_i, an arbitrary combination of tuples in $R(x_i)$ constitutes a possible world instance (*PW*), all possible world instances constitute possible world of x_i. We denote the possibility of possible world instance by *PPW*, can be computed as follows:

$$PPW = \prod_{xj \in R(xi) \wedge xj \in PW} pj \prod_{xk \in R(xi) \wedge xk \notin PW} (1 - pk) \tag{1}$$

Where x_j and x_k are objects in $R(x_i)$, x_j is in the possible world instance *PW* and x_k is not in *PW*, p_j and p_k are existential probability of x_j and x_k respectively.

When considering deterministic data, a distance-based outlier is a data who has not enough neighbors within a given distance threshold [15]. For uncertain data, however, existential probability needs to be taken into account in detection. We can only judge whether an object is an outlier through calculating the probability of the object to become an outlier.

Definition 3 (Outlier probability). For an uncertain data x_i, we define the sum of the probabilities of possible world instances that contain less than n tuples as its outlier probability. Then,

$$P_{outlier(xi)} = \sum_{PW_j \in Sn(xi,PW)} PPWj \tag{2}$$

Where $P_{outlier(xi)}$ denotes the outlier probability of x_i. $S_n(x_i, PW)$ is the set of possible world instances that contain less than n tuples in possible world of x_i.

Definition 4 (Top-k Outlier). We sort all uncertain data in descending order of their outlier probability, and collect the preceding k objects as top-k outliers on uncertain data.

2.2 Dynamic Programming Algorithm

The key computation of top-k outlier detection is calculating outlier probability of each data. The basic method is to list all possible world instances, and calculates the sum of the probabilities of possible world instances that contain less than n tuples. But the time complexity of this method shows exponential growth with neighborhood scale.

A dynamic programming algorithm is proposed by Wang B in [12]. In this paper, we improve the dynamic programming algorithm to calculate outlier probability, which makes the time complexity of the probability calculation decrease to $O(nN)$ from $O(n \cdot 2^N)$.

If $R(x_i)$ is ud-neighborhood of uncertain data x_i, let $R(x_i) = \{x_0, x_1,\ldots,x_{m-1}\}$. $[R(x_i), j]$ denotes event that only occurs j tuples in $R(x_i)$. We find all events in $[R(x_i), j]$ are exactly possible world instances that contain j tuples in $R(x_i)$. According to this, we can use a meaningful equivalent conversion for computing the outlier probability of an object. We list all the possible events. Then,

$$[R(xi) < n] = [R(xi), 0] \cup [R(xi), 1] \cup \cdots \cup [R(xi), n-1] \tag{3}$$

Where $[R(x_i) < \mathrm{n}]$ denotes all events that occur less than n tuples in $R(x_i)$, $[R(x_i), j]$ (j = 0, 1, … ,n-1) denotes the event that only occurs j tuples in $R(x_i)$.

Then, the outlier probability of x_i can be converted to another expression:

$$\begin{aligned} P_{outlier(xi)} &= P[R(xi) < n] = P[R(xi), 0] + P[R(xi), 1] + \cdots + P[R(xi), n-1] \\ &= \sum_{j=0}^{n-1} P[R(xi), j] \end{aligned} \tag{4}$$

Where $P[R(x_i) < n]$ denotes the probability of event $[R(x_i) < \mathrm{n}]$. $P[R(x_i), j]$ (j = 0, 1, … ,n-1) denotes the probability of event $[R(x_i), j]$.

The problem becomes how to calculate $P[R(x_i), j]$ efficiently, $R(x_i)$ will be divided into two parts based on the dynamic programming theory: the last tuple and the rest of tuples. If the last tuple occurs, the next step is to calculate probability of the event that only occurs *j-1* tuples in the rest of tuples; if the last tuple doesn't occur, the next step is to calculate probability of the event that occurs j tuples in the rest of tuples.

The order of tuples in $R(x_i)$ remains unchanged during the calculation. $|R(x_i)| = m$, probabilities of tuples in $R(x_i)$ are represented $p_0, p_1, \ldots, p_{m-1}$ respectively. In this paper, we use two-dimensional array to store the value of $P[R(x_i),j]$. We need to create a two-dimensional array T that contains m rows and n columns, T[i][j] denotes the probability of event that only occurs j tuples in the dataset that consisted by the first i tuples of $R(x_i)$. So $P[R(x_i) < \mathrm{n}]$ is the sum of values of the last row in T, then,

$$P_{outlier(xi)} = P[R(xi) < n] = \sum_{j=0}^{n-1} T[m][j] \tag{5}$$

The row number of array starts with 1 because it is meaningless when the formula $i = 0$. Solving formulas of two-dimensional array are as follows:

$$T[i][j] = \begin{cases} \overline{p_0} & if\, j = 0, i = 1 \\ p_0 & if\, j = 1, i = 1 \\ \overline{p_{i-1}} * T[i-1][0] & if\, j = 0, i > 1 \\ p_{i-1} * \mathrm{T}[i-1][j-1] & if\, j = i, i > 1 \\ p_{i-1} * \mathrm{T}[i-1][j-1] + \overline{p_{i-1}} * \mathrm{T}[i-1][j] & if\ j \neq 0, j < i \\ 0 & if\ j > i \end{cases} \tag{6}$$

For instance, R_1 is ud-neighborhood of an uncertain object, let $R_1 = \{x_0, \ldots, x_{m1-2}, x_{m1-1}\}$. Figure 1 shows the storage situation in two-dimensional array when calculating $P[R_1 < \mathrm{n}]$ by formula(6).

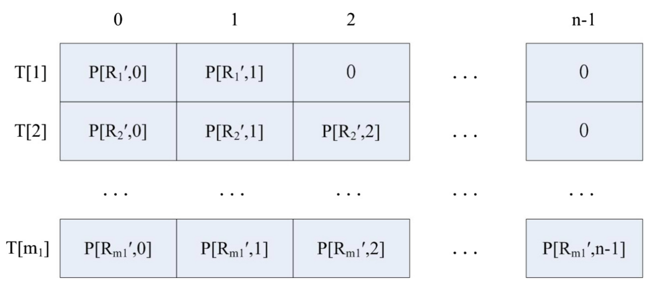

Fig. 1. Storage situation in two-dimensional array of R_1

Where R_i' (i = 1,…, m_1) is a data set that consists of the first i tuples appear of R_1. And $P[R_1 < n] = T[m_1][0] + \ldots + T[m_1][n-1]$.

In this paper, the above dynamic programming algorithm is represented by DPA.

2.3 Basic Algorithm

In this paper, k objects with maximum outlier probability in D are considered to be outliers. We need a data container to store k objects. Threshold δ has been always the minimum outlier probability in the data container. Considering that we need to find the minimum outlier probability in k objects every time, the minimum heap is used as the data container. The main idea of the algorithm is to search ud-neighborhood of each uncertain object x_i, and calculate the outlier probability of x_i by DPA, if the number of objects is less than k in the minimum heap, then insert x_i into the minimum heap, otherwise, assign outlier probability of the top object of the minimum heap to δ and replace the top object of the minimum heap with x_i if its outlier probability is greater than δ. The pseudo code is as follows:

```
Input: D, ud, n, k
Output: top-k outlier
1 δ=∞
2 for there exists at least one uncertain data $x_i$,$x_i \in$D
    Search for $R(x_i)$ ;
    Calculate $P_{outlier(xi)}$ by DPA;
    If (|minheap|<k)
      minheap.insert($x_i$);
    Else
      δ=outlier probability of the top object;
      If $P_{outlier(xi)}$>δ
        minheap.erase(top object);
        minheap.insert($x_i$);
      End if
    End if
  End for
3 output objects in minimum heap.
```

The above algorithm is called basic algorithm of top-k outlier detection or BA.

In the experiment, we find that BA needs to search ud-neighborhood and calculate the outlier probability for each object, which inevitably brings high time complexity. Then a virtual grid-based optimization algorithm is proposed to reduce the consumption of the algorithm and optimize BA.

3 Virtual Grid-Based Optimization Approach

3.1 Dynamic Increasing Calculation

Through further study on DPA, we find the following properties:

Property1: Outlier probability isn't affected by the order of tuples in $R(x_i)$.

Property2: Given two uncertain datasets U and U′ (|U| > n and |U′| > n), $P[U' < n] \geq P[U < n]$ holds, if $U' \subseteq U$.

Proof. We use $[U \geq n]$ to denote the event that at least n elements appear in U. Since $U' \subseteq U$, we get $U = U' \cup (U - U')$. Therefore, $[U' \geq n] \square [\mathrm{U} - U' \geq 0]$ is a sub-event of $[U \geq n]$, Since elements are mutually independent, thus $P[U' \geq n] * P[U - U' \geq 0] \leq P[U \geq n]$

, where $[U - U' \geq 0]$ is a certain event and $P[U - U' \geq 0] = 1$.Therefore, $P[U' \geq n] \leq P[U \geq n]$

holds and $1 - P[U' \geq n] \geq 1 - P[U \geq n]$.Therefore, $P[U' < n] \geq P[U < n]$.

According to the above properties, For uncertain data x_i, R' is a partition of $R(x_i)$, x_i isn't an outlier if the value of $P[R' < \mathrm{n}]$ is less than or equal to the threshold δ. So we needn't find all ud-neighborhood of x_i when detecting x_i. A pruning rule is proposed out of such point view in the next section.

In the process of judging whether uncertain data x_i is an outlier, we need to calculate $P[R' < \mathrm{n}]$ many times. If we calculate $P[R' < \mathrm{n}]$ by starting all over again, time cost will be undoubtedly huge. So we propose the dynamic increasing calculation, which doesn't increase the total time regardless of the times to calculate when detecting an object.

If R_2 is an intersection of R_1 (Sect. 2.2) and $\{x_0\prime, x_1\prime, \ldots, x_{m2-1}\prime\}$ ($R_2 = \{x_0, \ldots, x_{m_1-2}, x_{m_1-1}, x'_0, x'_1, \ldots x'_{m_2-1}\}$). Figure 2 shows the storage situation in array when calculating $P[R_2 < \mathrm{n}]$.

Through comparison between Fig. 1 and Fig. 2, we don't need to calculate $P[R_2 < \mathrm{n}]$ again from the very beginning, because the value of the first m_1 rows of Fig. 2 and Fig. 1 is completely identical, so we just need to calculate from $(\mathrm{m}_1 + 1)_{\mathrm{th}}$ row to $(\mathrm{m}_1 + \mathrm{m}_2)_{\mathrm{th}}$ row.

The above method is called dynamic increasing calculation (DIC) which is proposed to reduce the amount of calculation of the pruning algorithm when used as an auxiliary algorithm in the second stage in the next section.

	0	1	2	...	n-1
T[1]	$P[R_1',0]$	$P[R_1',1]$	0	...	0
T[2]	$P[R_2',0]$	$P[R_2',1]$	$P[R_2',2]$	...	0
	...	...	...	...	...
$T[m_1]$	$P[R_{m1}',0]$	$P[R_{m1}',1]$	$P[R_{m1}',2]$	...	$P[R_{m1}',n-1]$
	...	...	...	...	...
$T[m_1+m_2]$	$P[R_{m1+m2}',0]$	$P[R_{m1+m2}',1]$	$P[R_{m1+m2}',2]$	...	$P[R_{m1+m2}',n-1]$

Fig. 2. Storage situation in two-dimensional array of R_2

3.2 Virtual Grid-Based Optimization Algorithm

Wang B proposed a pruning strategy based on grid in [12], and the transformation is speeded up. However, in the large data sets, the data distribution is relatively sparse, empty cells account for a large proportion of the grid, which wastes a lot of storage space and access time. In this paper, we introduce a virtual grid structure [16] that only stores nonempty cell to effectively avoid the happening of this kind of situation.

In this section, the outlier detection process is divided into two stages. In the first stage, dataset is simply clustered. In the second stage, detecting outlier for clustering results by pruning rule.

(1) The First Stage. In the first stage, we realize relatively simple cluster analysis for uncertain dataset. We needn't to seek the optimal clustering effect, the key is efficiency of the algorithm. In this paper, we only discuss and analyze uncertain data in a two-dimensional space.

(1) Virtual grid structure

In order to avoid storing and searching empty cells, this paper introduces the virtual grid structure (VG).

Cell structure: each cell structure consists of 5-tuple < *X, checked, info, down, right* > , as shown in Fig. 3.

X	checked	info	down	right

Fig. 3. Cell structure

Where *X* is the coordinate value of the cell; *checked* denotes whether the cell is checked; *info* denotes information of tuples in the cell, such as tuple set, the number of tuples, the sum of probabilities of tuples; *down* point to the next cell in the same column; *right* point to the next cell in the same row.

For a given uncertain dataset D, we divide its domain into many mutual disjoint square cells and calculate the number of cells in each dimension. According to the number of cells in each dimension, we establish a total head node and two head nodes. Sequentially reading tuple x_i from uncertain dataset D, we add x_i into the cell and update information of the cell if the cell that contains x_i is already in VG; otherwise, we need to create a new cell node and insert it into cross list, then add x_i into the cell and update information of the cell. VG don't finish until all tuples are read. Its structure is shown in Fig. 4.

Each side of the cell is ud/2. Let $C_{x,y}$ be a cell of 2-D space. $L(C_{x,y}) = \{C_{u,v}|\text{u} = \text{x}\pm2, \text{v} = \text{y}\pm2, C_{u,v} \neq C_{x,y}\}$ denotes the neighbor cell set of $C_{x,y}$. Cell holds the following natures: for $\forall xi \in C_{x,y}$, there is $R(x_i)\in(C_{x,y}\cup L(C_{x,y}))$, and $C_{x,y}\in R(x_i)$. So we only need to find ud-neighborhood of x_i in $L(C_{x,y})$ when searching $R(x_i)$.

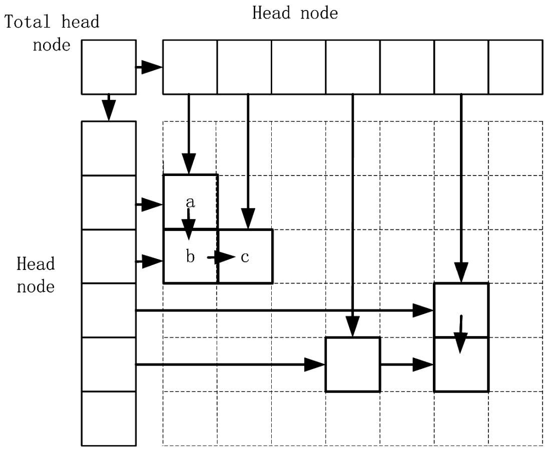

Fig. 4. The structure of VG

(2) Clustering

Once VG is constructed, we traverse the VG, and get a set of adjacent cells in the first column (such as a and b in Fig. 4), then store them into a temporary list and cluster set, then find adjacent cell in the same row (such as b and c in Fig. 4) of them and store adjacent cell which does not exist in temporary list into cluster set, finally clear temporary list. Repeat above steps until there is no cell in VG.

$LC = \{C_1,\ldots, C_i,\ldots, C_M\}$ denotes the set of clusters after clustering. Let $|C_i|$ is the number of tuples in C_i, $Count_i$ is the number of cells in C_i. According to the properties of the DPA, the outlier probability of an uncertain data is influenced by probability distribution and tuple density. The probability of containing outliers in C_i is larger when its sum of existential probability is smaller and the number of cells in C_i is larger. Considering the aforementioned factors, we measure the probability of containing outliers in C_i by the average probability of the cells in C_i. Let *den* be probability threshold. Then,

$$den = \frac{\sum_{j=1}^{|C_i|} p_j}{Count_i} \tag{7}$$

Where p_j is the existential probability of the object in C_i. The smaller *den* is, the greater the probability of containing outliers in C_i becomes.

In the process of the algorithm, δ is gradually increasing, the greater δ is, the more objects can be pruned. So, optimization algorithm prioritizes clusters whose *den* are minimum, then cells in the cluster are sorted by the sum of probabilities in ascending order, and detecting outlier from cell whose sum of probabilities is minimum, which makes δ rapidly increase.

(2) The Second Stage. In the second stage, we need to detect outlier for each cluster based on pruning rule.

In the process of neighbor search or calculation of a data, we can immediately interrupt the search or calculation if we can judge the non-outlier as early as possible. So the following pruning rule is presented. M is a dataset that stores neighbors of the object in the process of the algorithm.

Pruning rule: If $P[M < n] \leq \delta$, and M only contains part of the neighborhood of a query object, this query object can be pruned as non-outlier. A special case: If $P[M < n] \leq \delta$, and M only contains all objects of a cell, all objects in this cell can be pruned as non-outlier.

In the process of judging whether uncertain data x_i is an outlier, firstly, if the number of objects in the minimum heap is less than k, then calculate outlier probability of x_i and insert x_i into the minimum heap. Otherwise, all tuples in the cell that contains x_i are stored into M, then we calculate $P[M < n]$, if $P[M < n] \leq \delta$, all tuples in this cell can be pruned as non-outlier, otherwise, find a neighbor cell in the cluster that contains x_i, and search neighbors in the neighbor cell and store them into M, then calculate $P[M < n]$ (by DIC) when a cell is finished. If $P[M < n] \leq \delta$, x_i is not an outlier. If it still does not meet the pruning condition and has undetected neighbor cells when all neighbor cells in the cluster that contains x_i are detected, we need to find neighbor cells in VG, repeat above calculation and judgment. If all neighbor cells are evaluated, $P[M < n]$ is still greater than δ, then remove the top object in minimum heap and insert x_i into minimum heap, and outlier probability of the top object in minimum heap is assigned to δ, continue to test the next object.

The whole algorithm flowchart is shown in Fig. 5.

In the process of the algorithm, δ is gradually increasing, the vast majority of objects only need to search a small part of ud-neighborhood, which can judge whether the object is an outlier, thus save a lot of time.

Neighbor cells of the vast majority of objects are practically clustered in a same cluster, and only a few objects need to search VG when searching their ud-neighborhood in the first stage of the optimization algorithm.

In this paper, the above algorithm is called top-k virtual grid-based outlier detection algorithm on uncertain data (VGUOD for short).

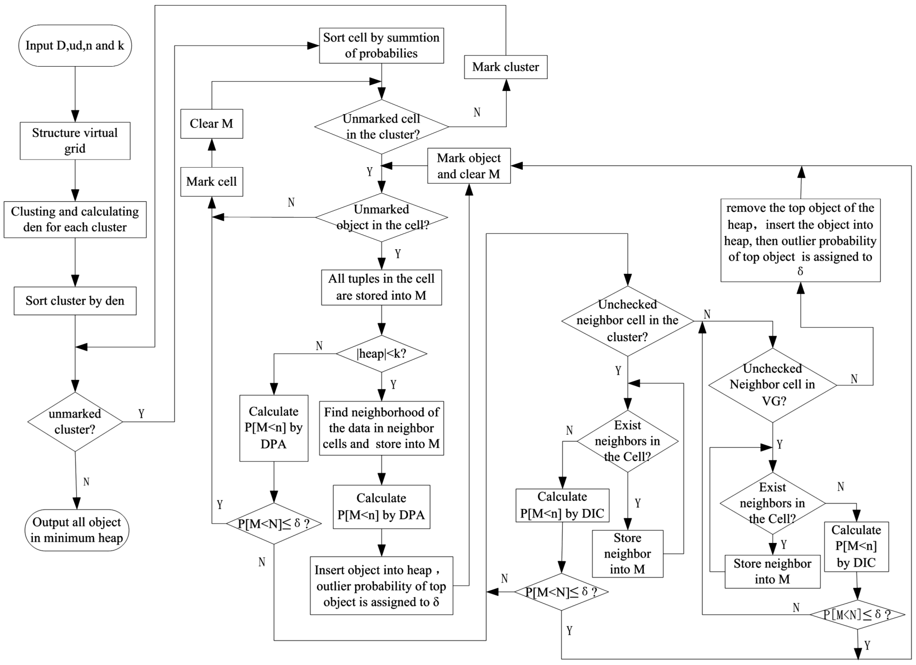
Input D,ud,n and k
Structure virtual grid
Clusting and calculating den for each cluster
Sort cluster by den
unmarked cluster?
Output all object in minimum heap
Sort cell by summtion of probabilies
Mark cluster
Clear M
Mark cell
Unmarked cell in the cluster?
Mark object and clear M
Unmarked object in the cell?
All tuples in the cell are stored into M
|heap|<k?
Calculate P[M<n] by DPA
Find neighborhood of the data in neighbor cells and store into M
Calculate P[M<n] by DPA
P[M<N]≤ δ ?
Insert object into heap, outlier probability of top object is assigned to δ
Unchecked neighbor cell in the cluster?
Exist neighbors in the Cell?
Calculate P[M<n] by DIC
Store neighbor into M
P[M<N]≤ δ ?
remove the top object of the heap, insert the object into heap, then outlier probability of top object is assigned to δ
Unchecked Neighbor cell in VG?
Exist neighbors in the Cell?
Store neighbor into M
Calculate P[M<n] by DIC
P[M<N]≤ δ ?
Y
N

Fig. 5. Algorithm flowchart

4 Experiments

To evaluate the performance of the proposed algorithm objectively, in this chapter, the experiment is taken on two synthetic datasets SynDS1 and SynDS2 and mainly focuses on the influence of several parameters and comparisons in precision and running time between VGUOD and GPA that proposed in [12]. Software environment: 2.93GHZ CPU、2G main memory and Microsoft Windows7 Ultimate system. Experiment platform: Microsoft Visual Studio 2010. Language: C ++.

Synthetic datasets are generated by Matlab. Each dataset has 100000 uncertain records. Valued attributes are 2-dimensional and each dimension is floating point number that distribute in [0, 1000], SynDS1 and SynDS2 are composed respectively of several normal distributions and uniform distributions. Existential probability was randomly generated in the range of (0, 1).

4.1 Influence from Parameters

In order to analyze the influence of parameters on the performance of the algorithms, we evaluate the effectiveness of BA and VGUOD by using SynDS1. We need to set values for parameters before the experiments begin. Let $n = 4$, $ud = 20$, $k = 0.03*N$, $|D| = N$.

Firstly, we discussed the effect of the parameter *k*, we varied *k* from *0.01*N* to *0.05*N* with increment *0.01*N* while keeping other parameters constant, then recorded the running time. The running time of BA and VGUOD is shown in Fig. 6.

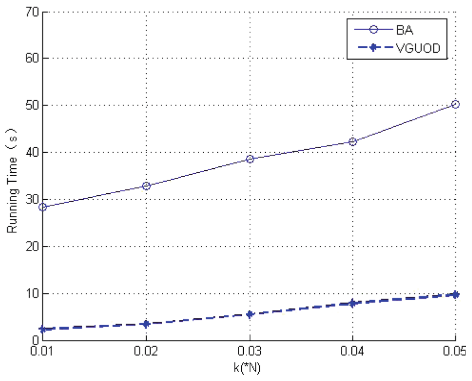

Fig. 6. Running time vs. k

The number of outliers was increasing as *k* grew, and running time of two approaches is increasing. However, as the Fig. 6 shows, the running time of VGUOD is far less than running time of BA, because virtual grid structure can filter all empty cells and find neighbors more easily, besides, pruning method based on virtual grid structure can trim most non-outliers, so VGUOD can effectively save running time.

The relationship between *ud* and running time is illustrated in Fig. 7. The number of ud-neighborhood of an object for different *ud* (*ud* is increasing from 10 to 28) is increasing. BA needs to search the whole dataset and calculates the outlier probability for each object, which inevitably costs more computation time. However, for VGUOD, the value of $P[M < n]$ in a cell is declining as *ud* grow. The smaller the value of P $[M < n]$ is, the greater the probability of meeting the pruning condition becomes. So VGUOD needs less running time than BA.

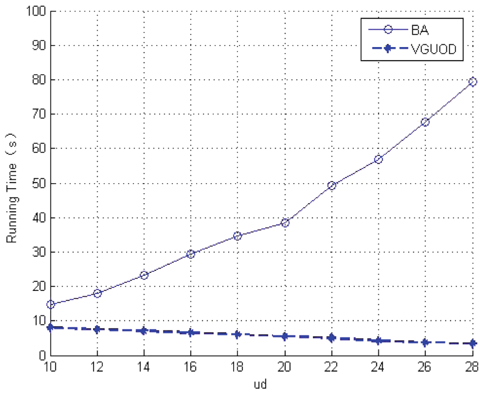

Fig. 7. Running time vs. ud

Then we analyzed the influence of parameter n on running time. The parameter n varied from 4 to 20 to test BA and VGUOD. The cost of calculating outlier probability is increasing as n gets larger, which increases running time. Since BA is running without any pruning strategy, it spends more time calculating outlier probability of all objects in dataset. VGUOD effectively reduces the impact of parameter n on running time by using pruning rule and DIC. As Fig. 8 shows, the running time of VGUOD is far less than running time of BA.

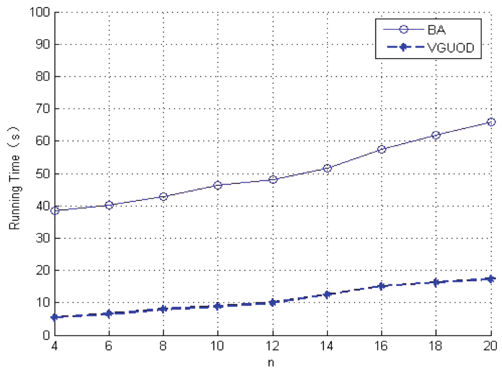

Fig. 8. Running time vs. n

Finally, the effects of the change of data size on the running time were discussed. We used ten datasets that generated by Matlab and the number of records of them varied from 20000 to 160000 to test BA and VGUOD. Figure 9. shows both the running time of BA and VGUOD in different datasets. The more the number of records in dataset, the more the amount of calculation and the running time consumption. BA needs to calculate all objects, its computational effort remarkably increases with the size of the dataset. The running time of VGUOD is far less than the time of BA because of its pruning ability.

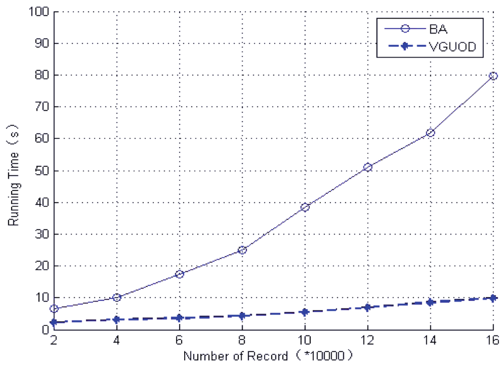

Fig. 9. Running time vs. Number of Record

4.2 Compared with Other Algorithm

In this section, we evaluate the precision and execution time cost of VGUOD compared with GPA. We use the ratio of the right number of outliers found by the algorithm to the total numbers of outliers found by the algorithm to evaluate the precision of the algorithm. Apparently, the higher the ratio, the higher the precision of the algorithm. So we run the GPA algorithm and the VGUOD algorithm on SynDS1 and SynDS2 and calculate the ratio Z by Eq. (8).

$$Z = \frac{b}{B} \tag{8}$$

Where B denotes the total number of outliers found by the algorithm, and b is the right number of outliers found by the algorithm.

Figures 10 and 11 respectively contrasts the precision and execution time cost of GPA and VGUOD when they run in the same dataset to detect the same number of outliers. Let *ud = 20, n = 4*.

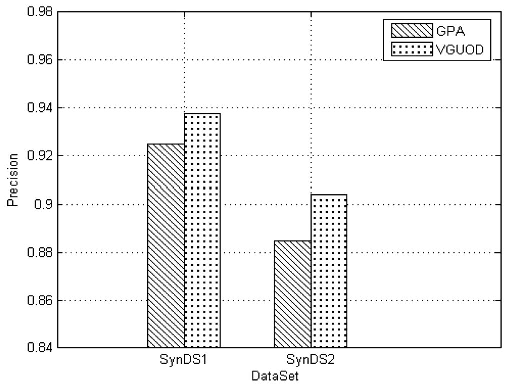

Fig. 10. Precision

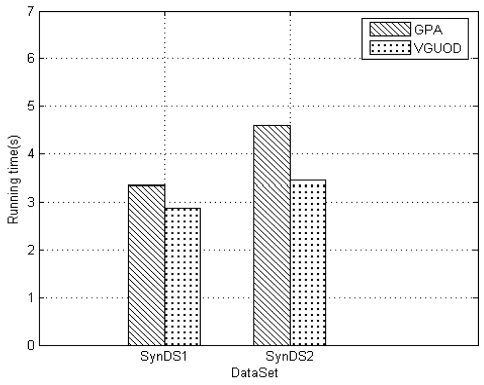

Fig. 11. Running time

By analyzing experimental results, we can observe that the VGUOD has advanced performance in both running time and precision than GPA. VGUOD algorithm gets top-k outliers, which guarantees it has a higher precision than GPA when detecting the same number of outliers. In terms of running time, VGUOD algorithm filters empty cells and judges whether an object is outlier by using less computation amounts, which greatly saves running time.

5 Conclusions

As a new but important research field, Outlier detection on uncertain data has a good extensive application prospect. In this paper, a new definition of outlier detection on uncertain data is put forward, and then we introduce the dynamic programming idea to efficiently calculate the outlier probability of the data, and propose an efficient virtual grid-based optimization method. The algorithm adapts to detect outliers in large dataset to a certain extent.

We will study more complex uncertain data model in the future work, and detecting outlier on uncertain data in high dimensional data space.

References

1. Zhang, J., Zulkernine, M.: Anomaly based network intrusion detection with unsupervised outlier detection. In: IEEE International Conference on Communications, ICC 2006, pp. 2388–2393. IEEE (2006)
2. Alaydie, N., Fotouhi, F., Reddy, C.K., Soltanian-Zadeh, H.: Noise and outlier filtering in heterogeneous medical data sources. In: 2012 23rd International Workshop on Database and Expert Systems Applications, pp. 115–119. IEEE (2010)
3. Knorr, E.M., Ng, R.T.: Algorithms for mining distance-based outliers in large datasets. In: Proceedings of the 24rd International Conference on Very Large Data Bases, pp. 392–403. Morgan Kaufmann Publishers Inc. (1998)
4. Wang, L., Zou, L.: Research on algorithms for mining distance-based outliers. J. Electron. **14**, 485–490 (2005)
5. Han, J., Kamber, M.: Data Mining–Concepts and Techniques 2nd ed. Data Mining Concepts Models Methods & Algorithms Second Edition 10(9),1–18 (2006)
6. Knorr, E.M., Ng, R.T.: Finding intensional knowledge of distance-based outliers. In: VLDB, pp. 211–222 (1999)
7. Knorr, E.M., Ng, R.T., Tucakov, V.: Distance-based outliers: algorithms and applications. VLDB J. — Int. J. Very Large Data Bases **8**(3–4), 237–253 (2000)
8. Aggarwal, C.C., Yu, P.S.: Outlier detection with uncertain data. In: SDM (2008)
9. Shaikh, S.A., Kitagawa, H.: Distance-based outlier detection on uncertain data of Gaussian distribution. In: Sheng, Q.Z., Wang, G., Jensen, C.S., Xu, G. (eds.) APWeb 2012. LNCS, vol. 7235, pp. 109–121. Springer, Heidelberg (2012)
10. Shaikh, S.A., Kitagawa, H.: Fast top-*k* distance-based outlier detection on uncertain data. In: Wang, J., Xiong, H., Ishikawa, Y., Xu, J., Zhou, J. (eds.) WAIM 2013. LNCS, vol. 7923, pp. 301–313. Springer, Heidelberg (2013)

11. Shaikh, S.A., Kitagawa, H.: Top-k outlier detection from uncertain data. Int. J. Autom. Comput. **11**(2), 128–142 (2014)
12. Wang, B., Xiao, G., Yu, H., et al.: Distance-based outlier detection on uncertain data. In: IEEE Ninth International Conference on Computer & Information Technology, pp. 293–298. IEEE (2009)
13. Wang, B., Yang, X.-C., Wang, G.-R., Ge, Yu.: Outlier detection over sliding windows for probabilistic data streams. J. Comput. Sci. Technol. **25**(3), 389–400 (2010)
14. Abiteboul, S., Kanellakis, P., Grahne, G.: On the representation and querying of sets of possible worlds. In: PODS 2001, pp. 34–48 (1991)
15. Angiulli, F., Pizzuti, C.: Fast outlier detection in high dimensional spaces. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) PKDD 2002. LNCS (LNAI), vol. 2431, pp. 15–27. Springer, Heidelberg (2002)
16. Dong, J., Cao, M., Huang, G., Ren, J.: Virtual grid-based clustering of uncertain data on vulnerability database. J. Convergence Inf. Technol. **7**(20), 429–438 (2012)