Mining Outliers in Spatial Networks

Wen Jin[1], Yuelong Jiang[1], Weining Qian[2], and Anthony K.H.Tung[3]

[1] School of Computing Science,Simon Fraser University
{wjin, yjiang}@cs.sfu.ca
[2] Department of Computer Science, Fudan University
wnqian@cs.fudan.edu.cn
Department of Computer Science, National University of Singapore
atung@comp.nus.edu.sg

Abstract. Outlier analysis is an important task in data mining and has attracted much attention in both research and applications. Previous work on outlier detection involves different types of databases such as spatial databases, time series databases, biomedical databases, etc. However, few of the existing studies have considered spatial networks where points reside on every edge. In this paper, we study the interesting problem of distance-based outliers in spatial networks. We propose an efficient mining method which partitions each edge of a spatial network into a set of length d segments, then quickly identifies the outliers in the remaining edges after pruning those unnecessary edges which cannot contain outliers. We also present algorithms that can be applied when the spatial network is updating points or the input parameters of outlier measures are changed. The experimental results verify the scalability and efficiency of our proposed methods.

1 Introduction

Outlier analysis, which aims to find a small number of exceptional objects in a database, is an important data mining task. Methods have been developed for different types of databases such as spatial databases [1, 2, 13, 17, 16, 19], time series databases [10, 14], biomedical databases [21, 22], etc. In these methods, the databases are typically considered static and relational, and the distance used in identifying outliers is always measured by Euclidean distance.

However, in many real applications where spatial data are managed, *spatial objects are often added or removed, and the position and accessibility of spatial objects are constrained by spatial networks* [21]. Examples include road networks, river networks, plane networks, rail networks, etc.

In general, a spatial network can be modeled as a graph where points are located on the edges. These points can be static objects such as buildings, or snapshots of mobile objects such as vehicles. Clearly, the actual distance between any two points, called *network distance*, is measured by the length of the shortest path connecting them in the network instead of Euclidean distance. Figure 1 depicts an example of a spatial network, where each node is denoted by a square,

each edge is associated with a distance label, and each point (object) is denoted by a cross and lies exactly on an edge.

Mining outlying objects in a spatial network can provide very useful knowledge to decision makers. For example, a road supervisor is interested to know which(or how many) vehicles are most deviant to other vehicles in distance during rush hour, or which roads or portions of a road have the least traffic. Such outlier information can be used to analyze traffic patterns in a network and help adjust the coverage of traffic volume.

Due to the features of shortest distance and the dynamics of spatial networks, state-of-the-art algorithms on mining outliers in traditional databases cannot be straightforwardly applied in spatial networks. As shown in Figure 1, point P_{12}, P_{19} and P_{26} are close to each other in Euclidean distance(if not considering network constraint). For instance, if we apply the well-known cell-based outlier detection method [12] to this graph, P_{12}, P_{19} and P_{26} will be grouped into a small cell, and will be treated as non-outliers. However, since the only route to P_{26} in the network is through the edge (n_7, n_4), and the remaining points have much larger network distances to P_{26} than to any other point, so P_{26} is an outlier. This problem motivates us to design a novel method to find outliers in a spatial network efficiently. Some recent work has been done on identifying graph outliers [4, 8, 20], but the processed graph employed contains only nodes and edges, and is different from a spatial network where additional points reside on the edges. Furthermore, these existing methods basically aim to find exceptional nodes or exceptional edges.

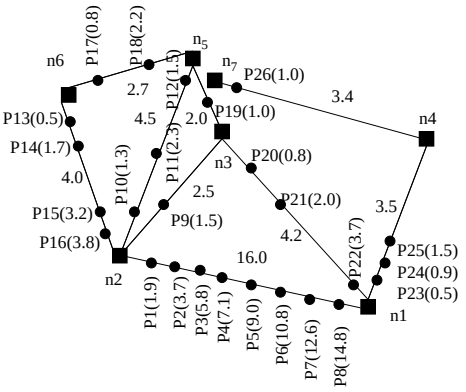

Fig. 1. An example of a spatial network

In this paper, we study the interesting problem of distance-based outliers in spatial networks. We propose an efficient mining method which partitions each edge of a spatial network into a set of length d segments, then quickly identifies the outliers in the remaining edges after pruning those unnecessary edges which cannot contain outliers. We also present algorithms for dynamical settings where points in a spatial network are inserted/removed, or where the input parameters of outlier measures are changed. Our contributions are as follows:

- We introduce the problem of distance-based outliers in spatial networks.
- We develop very efficient algorithms for mining outliers in spatial networks in both static settings and dynamic settings, including when points in a spatial network or input parameters of outlier measures are updated.
- We perform extensive experiments on both synthetic datasets and real datasets.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 introduces the preliminaries of outliers in spatial networks. Section 4 introduces the mining algorithm for static spatial networks. Section 5 introduces the mining algorithm for spatial networks in dynamic settings. Section 6 presents the performance analysis of these methods. We conclude the paper with a summary in Section 7.

2 Related Work

In the database point of view, recent research on outlier detection can be categorized into statistics-based, distance-based, density-based and clustering-based approaches [9].

Outlier research has its roots in statistics [7, 3], and this early work can be classified as distribution-based and depth-based. A distribution-based method uses some kind of data distribution model such as Normal, which is mostly univariate, to describe the properties of a dataset. It then tests for outliers based on the postulated distribution. The problem of this method is that it assumes that the dataset possesses some probability distribution beforehand. In real applications, it is difficult to know the underlying data distribution. A depth-based method uses computational statistics to represent data in different depths and outliers are probably those data in lower depths. However, as such a method relies on k-dimensional convex hulls computation with a lower bound cost of $\Omega(n^{k/2})$, it is not efficient for high dimensions.

The concept of distance-based outliers, proposed by Knorr and Ng [12], defines an object p being an outlier, if at most n objects are within distance d of p. Outliers pertain to the global view of a dataset. A cell-based outlier detection approach that partitions a dataset into cells is also presented in the work. The time complexity of this cell-based algorithm is $O(N + c^k)$ where k is dimension number, N is dataset size, and c is a number inversely proportional to d. For very large databases, this method achieves better performance than depth-based methods. However, it is still exponential to the number of dimensions. Ramaswamy et al. extended the notion of distance-based outliers by using distance to the k-nearest neighbor to rank outliers [19], where an efficient algorithm is given based on the technique of partitioning dataset and distance bounding [18].

Some clustering algorithms such as CLARANS [15], DBSCAN [5], BIRCH [25], and CURE [6] consider outliers, but only to the point of ensuring that they do not interfere with the clustering process. Further, outliers are only by-products in clustering algorithms, and generally, clustering algorithms cannot be applied directly to a spatial network to find outliers.

Breunig et al. introduced the concept of local outliers, which assigns each piece of data a local outlier factor (LOF) of being an outlier, depending on its neighborhood[2]. This outlier factor can be used to rank objects according to their outlierness. Computing the LOF of all objects in a database requires O(n*runtime of a knn query). The outlier factors can be computed very effi-

ciently if OPTICS is used to analyze the clustering structure. A top-n based local outliers mining algorithm which uses the distance bound of a micro-cluster to estimate density was presented in [11].

3 Preliminaries

Before we introduce mining algorithms for the spatial network outliers. Let us revisit related concepts and notions of spatial networks and outliers, interesting readers can see the details in [23, 12]. Let a spatial network be a weighted graph $G = (V, E, W)$, and V is a set of nodes, and E is a set of edges. The function $W : E \rightarrow R^+$ associates each edge with a positive weight. Without loss of generality, W can be regarded as the distance in an edge. The position of a *point* (i.e. object) p in the network can be expressed by $< n_i, n_j, pos >$ where the $pos \in [0, W(e)]$ and $e = (n_i, n_j)$. It shows p has *pos* unit away from the node n_i along the edge (n_i, n_j). As shown in Figure 1, P_{15} lies on the edge (n_6,n_2) and it is 3.2 units away from n_6 along the edge, so it is represented by $< n_6, n_2, 3.2 >$. We assume the number of points in the spatial network G is N, and the edges in the spatial network satisfies the triangle inequality.

Definition 1. *Let p and q be two points whose positions are (n_a, n_b, pos_p) and (n'_a, n'_b, pos_q), respectively. The* **direct distance** *$ddist(p, q)$ between points p and q is defined by $|pos_p - pos_q|$ if $n_a = n'_a$ and $n_b = n'_b$ (i.e., p and q lie on the same edge); otherwise, it is defined as ∞. Given a point p with position (n_a, n_b, pos_p), the direct distance $ddist(p, n_a)$ between p and n_a is pos_p. The direct distance $ddist(p, n_b)$ is defined by $W(n_a, n_b) - pos_p$[23].*

Note that *ddist* only works for two points lying on the same edge, while the *ddist* of a point from a node works only when the point is lying on an edge adjacent to the node.

Definition 2. *Given nodes n_i and n_j, the* **network distance** *$ndist(n_i, n_j)$ is defined as the distance of the shortest path from n_i to n_j and vice versa[23].*

Definition 3. *Given points p and q, where p lies on the edge (n_a, n_b) and q lies on the edge (n'_a, n'_b), the* **network distance** *$ndist(p, q)$ is the distance of the shortest path from p to q. $ndist(p, q)$ is defined by $\min_{x \in \{a,b\}, y \in \{a',b'\}}(ddist(p, n_x) + ndist(n_x, n_y) + ddist(n_y, q))$ if p and q lie on different edges; otherwise, $ndist(p, q)$ is the minimum of the previous quantity and $ddist(p, q)$[23].*

Since the number of nodes in the spatial networks is much smaller than that of points, so the network distance between each pair of nodes n_i and n_j can be materialized with little cost and will be used frequently to speed up distance comparisons in the stage of outliers detection. To facilitate efficient access, the adjacency list and points are stored in two separate flat files indexed by B+-trees[24], as shown in Figure 2 representing the spatial network of Figure 1.

Given a collection of N points that lie on a network, we aim to find a small group of points according to the following criteria.

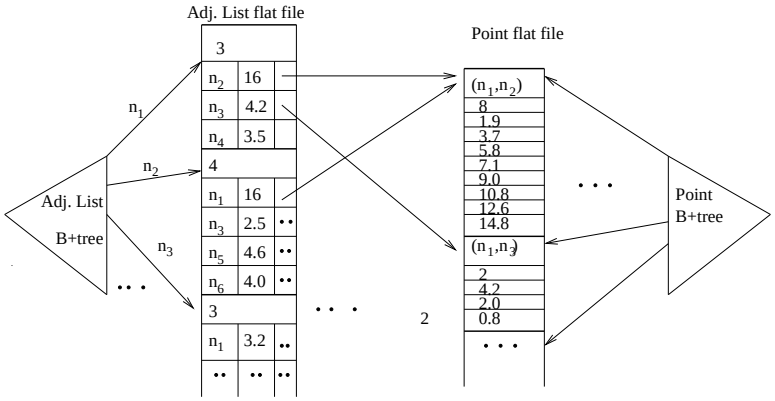

Fig. 2. Data structure of the network and segment trees

Definition 4. *Given user-defined parameters P and d, and a network distance function F, a point o in a spatial network G is a distance-based outlier if at least fraction P of points in G lie greater than* **network distance** *d from o.*

4 Mining Outliers in Static Spatial Networks

In this section, we first present a naive index-based method to identify the outliers among the points in network. Then we propose a novel edge-segmentation method which can significantly improve the performance of mining outliers.

4.1 A Naive Index-based method

For a point o in the spatial network G, the d-neighborhood of o contains the set of points that are within distance d of o. The fraction P is the minimum fraction of points in G that must be outside the d-neighborhood of an outlier. Obviously, given P and d, the problem of identifying distance-based outliers in the spatial network can be solved by answering a nearest neighbor search at each point o.

We can build an index by employing a range search with a network distance of d for each point o to find such outliers. If more than $(1-P)N$ points are found in the d-neighborhood, o is a non-outlier; otherwise, o is an outlier. When the d becomes larger, each range search costs larger which degrades the performance greatly, so the worst case of this method has the complexity of $O(N^2)$. Such an index can be maintained for answering outlier queries multiple times.

4.2 An edge-segmentation method

Before introducing our outliers mining algorithms, we first consider pre-processing a spatial network. We partition each edge $e = (n_i, n_j)$ into $\lceil \frac{w(e)}{d} \rceil$ segments

S_1 =$[n_i, n'_1)$, S_2 =$[n'_1, n'_2), \ldots, S_{\lceil \frac{w(e)}{d} \rceil -1}$ =$[n'_{\lceil \frac{w(e)}{d} \rceil -2}, n'_{\lceil \frac{w(e)}{d} \rceil -1}), S_{\lceil \frac{w(e)}{d} \rceil}$ =$[n'_{\lceil \frac{w(e)}{d} \rceil -1}$, $n_j]$ where each segment has a length of d, and n'_1, n'_2, …, $n'_{\lceil \frac{w(e)}{d} \rceil -1}$ are the positions of the segment ends. In practice, one end segment in each edge such as $S_{\lceil \frac{w(e)}{d} \rceil}$ may not be of a length of exactly d, but an actual length $d^* < d$. As shown in Figure 3, edge (n_1, n_2) is equally partitioned into segments [n_1, n'_1), [n'_1,n'_2), [n'_2,n'_3), [n'_3,n'_4) and [n'_4,n_2]. The statistics information such as the number of points in each segment is also maintained.

Now each edge segment of a spatial network can be categorized into one of the following types: (1) *Outlier Segment(OS)* which consists of outliers; (2) *Non-outlier Segment(NS)* which cannot contain any outlier; and (3) *Undetermined Segment(US)* which may or may not have outliers. Without any pairwise distance computation during the outlier detection, an *Outlier-Segments* returns objects that are outliers, or a *Non-outlier Segments* can be immediately pruned. The remaining outliers can be identified among the *Undetermined Segments*.

Now the problem is how to quickly identify the type of an edge as *Outlier-Segment*, *Non-outlier Segment* or *Undetermined Segment*.

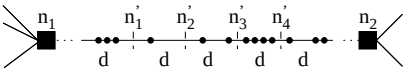

Fig. 3. Edge segmentation

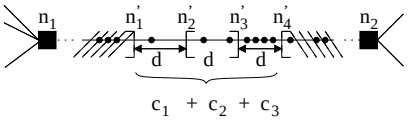

Fig. 4. OS and NS

Definition 5. *The smallest distance between segment* S_i =$[n_a$**,** $n_b]$ *and segment* S_j =$[n_c$**,**$n_d]$, **sdist(**S_i, S_j**)**, *is* $\min\{ndist(n_a, n_c)$, $ndist(n_a, n_d)$, $ndist(n_b, n_c)$, $ndist(n_b, n_d)\}$. *Segment* S_j *is* **adjacent** *to segment* S_i *if the smallest distance between* S_i *and* S_j, $sdist(S_i, S_j) < d$.

Here, S_j refers to the segment adjacent to S_i in either right side or left side. The adjacent segments of S_i are also called the d-neighborhood of S_i.

Definition 6. *The largest distance between segment* S_i =$[n_a$**,** $n_b]$ *and segment* S_j =$[n_c$**,**$n_d]$, **ldist(**S_i, S_j**)**, *is* $\max\{ndist(n_a, n_c)$, $ndist(n_a, n_d)$, $ndist(n_b, n_c)$, $ndist(n_b, n_d)\}$. *Segment* S_j *is* **complementary** *to segment* S_i *if the largest distance between* S_i *and* S_j, $ldist(S_i, S_j) \leq d$.

Here, S_i may also have multiple complementary segments. Specifically, if S_i is a segment with length d, there is no such complementary segment. If S_j is S_i's complementary segment, it must be S_i's adjacent segment; but if S_j is S_i's adjacent segment, it may not be S_i's complementary segment.

Lemma 1. *Given segment* S_i, S_j, *if* S_j *is not an adjacent segment to* S_i, *then any point* $p \in S_i$, $q \in S_j$ *must be more than* d *apart, i.e.* $ndist(p, q) > d$.

Lemma 2. *Given c_i points in segment S_i and c_j points in S_i's adjacent segments, S_i is an* **outlier segment (OS)** *if $c_i + c_j < (1-P)N$.*

Lemma 3. *Given c_i points in segment S_i and c_j points in S_i's complementary segments, S_i is a* **non-outlier segment(NS)** *if $c_i + c_j \geq (1-P)N$.*

As shown in Figure 4, the number of points in segments $[n_1', n_2']$, $[n_2', n_3']$, $[n_3', n_4']$ are c_1, c_2 and c_3 respectively. If $c_2 \geq (1-P)N$, none of points in $[n_2', n_3']$ is an outlier. If $c_1 + c_2 + c_3 < (1-P)N$, all the points in $[n_2', n_3']$ are outliers.

Those segments which are neither outlier segments nor non-outlier segments are **undetermined segments(US)**.

Thus, we can scan the spatial network once, and partitioning edges of the spatial network into segments of length d. Suppose there are **m** segments, and the whole points are categorized into three types of segments:**OS, NS** and **US**. Since the points in **OS** segments and **NS** segments can easily be identified as either outliers or non-outliers, the remaining mining task is obviously to further check those segments labeled as **"US"**. That is, each point p in such a **"US"** segment will be evaluated the distance *only* between the points in the d-neighborhood of the segment where p resides. If none of the points is identified as an outlier, the segment is labeled as **"NS"**, otherwise the segment is labeled as **"US(O)"**. The pseudo-code of the mining algorithm is as follows.

Algorithm 1 An Edge-Segmentation Outlier Detection Method.
Input: A spatial network $G = (V, E, W)$ partitioned into m segments, N, d, P
Output: Outliers in G
Method:

```
1.  FOR i = 1 to m DO Count_i = 0
2.  FOR each point p DO
3.      Map p to its segment S_i, increment Count_i by 1;
4.  FOR i = 1 to m DO
5.      IF Count_i + Σ_{S_j is complementary to S_i} Count_j ≥ (1-P)N THEN
6.          Label S_i as "NS";//S_i is a Non-Outlier Segment
7.      ELSE IF Count_i + Σ_{S_j is adjacent to S_i} Count_j < (1-P)N THEN
8.          Label S_i as "OS";//S_i is an Outlier Segment
9.      ELSE//S_i is an Undetermined Segment, needs to check its points one by one
10.             FOR each object p ∈ S_i DO
11.                 Count_P = Count_i;
12.                 FOR each object q ∈ S_j where S_j adjacent to S_i DO
13.                     IF ndist(p, q) ≤ d THEN
14.                         Increment Count_p by 1;
15.                 IF Count_p ≤ (1-P)N THEN
16.                     P is an outlier, label S_i as "US(O)";//S_i an outlier
17.                 IF S_i is not labeled as "US(O)" THEN
18.                     Label S_i as "NS";
19. Output outliers in G;
```

Step 1 takes m time(since there are m segments), where $m \ll N$ is the total number of segments. Step 2 to step 3 takes N time. For Step 4 to step 19, the worst case happens when each segment contains at most $N(1-P)$ objects, and each object in a segment is required to check up to $N(1-P)$ objects in each of the adjacent or complementary segments. As the number of adjacent or complementary segments is bounded by m, hence, step 4 to step 19 takes $O(m(N(1-P))^2)$ time. Thus the worst case time complexity is $O(m(N(1-P))^2 + N) = O(m(N(1-P))^2)$. Furthermore, based on the similar analysis in [12], we know that since P is expected to be extremely close to 1 in practice, especially for large datasets, so $O(mN^2(1-P)^2)$ can be approximated by $O(m)$, thus under such circumstance the time complexity is $O(m+N)$.

5 Mining Outliers in Dynamic Settings

In the previous section, we introduce an edge-segmentation method to efficiently mine distance-based outliers in a static setting in which the whole points are available in a spatial network. In this section, we will discuss the dynamic settings for outliers detection in the following two cases:(1) the points are added to or removed from edges, or (2) users may query the outliers with respect to different input parameters d or p. In both cases, it requires to effectively maintain the existing segmentation structure and mine the changes of outliers in an incremental way.

5.1 Mining outliers when points are updated

From the perspective of data management, points in the spatial network are always updated in the case of m_1 insertions or m_2 deletions. Since the distance-based outlier is dependent on N, so after updating, if the number of points is changed to N' where $N' = N + m_1 - m_2$, each point in the spatial network needs to check whether there are $(1-p)N'$ points within its d neighborhood region. Thus we can apply the edge-segmentation algorithm to the updated spatial network to find the changes of outliers. The problem is how to make full use of the existing segmentation structure and avoid unnecessary distance computation as much as it can?

It is clear that each segment is labeled either **OS, NS** or **US** after the outlier detection in static setting. Such information can incrementally maintained in the case of points deletion or insertion. Since the number of points is changed after updating points, the type of each segment S_i needs to be quickly updated based on Lemma 2 and Lemma 3 given in the previous section.

The problem is that if S_i is labeled as **OS** or **NS**, we obviously know that the whole points in S_i are either outliers or non-outliers, but for those **undetermined segments(US)** S_i, we still need further check every point P in S_i to see if it is an outlier or not. *To reduce the cost of such pairwise computation and to improve the efficiency of incremental mining as much as possible, we store the points in each edge into a binary tree, and the points of each segment can be*

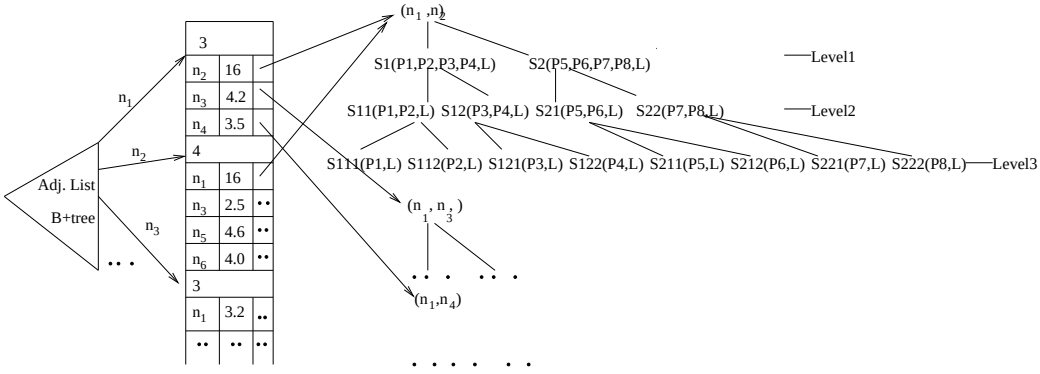

Fig. 5. Data structure of the network and segment trees

further partitioned till a length of u. For segment S_i, the points within it are organized as a binary tree. That is, if the parent node covers points in subsegment with length l, it will be expanded to two child nodes: the left child node covers points in left half of subsegment with length $l/2$ and the right child node covers points in right half. The tree expansion procedure starts with segment S_i (root node), and continues until one of following conditions is satisfied: (1)the node does not cover any points in current subsegment; (2) the length of subsegment is less than $2u$. Figure 5 shows the example of organizing edge segments ($d = 8$) and their subsegments corresponding to the network in Figure 1, into tree structures. The points of each segment are maintained into a binary segment tree. Each node in the tree is labeled (shown as "L" in Figure 5) in one of three types. We observe that the smaller the u, the larger size of the tree, and less number of points in each subsegment in the lowest level of the tree. As shown in the Figure 5, the tree only expands to the level 2 when $u = 4$, while it expands to the level 3(in dashed box) if $u = 2$.

Now we can **progressively check outlier points level-by-level instead of one by one**. If the current segment (subsegment) S_i is labeled as **US**, *the left and right child subsegments of* S_i *will be checked to see whether they are* **OS**, **NS** *or* **US**. Before checking Lemma 2 and Lemma 3, the number of points in current subsegment, its adjusted adjacent subsegments, and its adjusted complementary subsegments in same level of the trees should be obtained. Fortunately, all these information is stored in the trees and we do not need to access the points again. An example of progressive checking **US** segment is illustrated in Figure 6, where $d = 10$ and $u = 2$. Note that the length of segment S_{i3} is $d^* = 6$ since S_{i3} is the last segment in the edge.

If segment S_{i2} is undetermined, its subsegment S_{i21} and S_{i22} are checked. If subsegment S_{i21} is still undetermined, its subsegment S_{i211} and S_{i212} are further checked. For different segment S_{ik}, the range of its adjacent segments (shown in dashed lines) and the range of its complementary segments (shown in blacken lines) are adjusted correspondingly. We can see in Figure 6 that with the

decreasing of the subsegment length, the gap between these two ranges becomes smaller and smaller, so does the chance that the subsegment is still labeled as **NS**. The pseudo-code of the mining algorithm is as follows.

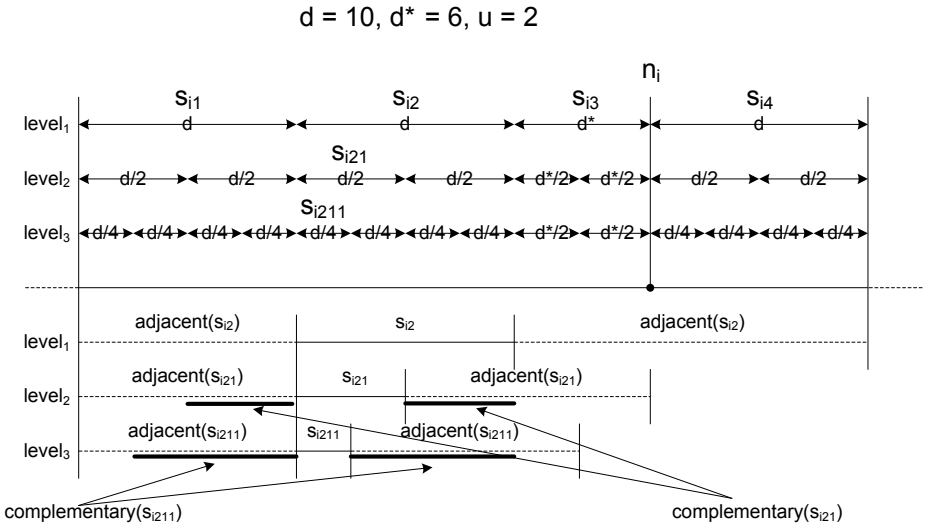

Fig. 6. Progressive check for "US" subsegments

Algorithm 2 A Dynamic Outlier Detection Method for Updated Points.
Input: Segment Trees of $G = (V, E, W)$, N, d, P, m_1 insertions, m_2 deletions
Output: Outliers in G
Method:

1. FOR $i = 1$ to $m_1 + m_2$ DO
2. Update points in G and counts in corresponding segments and subsegments;
3. FOR $i = 1$ to m DO
4. IF $Count_i + \sum_{S_j \, is \, complementary \, to \, S_i} Count_j \geq (1 - P)(N + m_1 - m_2)$ THEN
5. Label S_i as **"NS"**; //S_i *is a Non-Outlier Segment*
6. ELSE IF $Count_i + \sum_{S_j \, is \, adjacent \, to \, S_i} Count_j < (1 - P)(N + m_1 - m_2)$ THEN
7. Label S_i as **"OS"**; //S_i *is an Outlier Segment*
8. ELSE //S_i *is an Undetermined Segment,* **check its subsegments or points**
9. $SubsegmentCheck(S_i)$;
10. Output outliers in G;

Procedure: $SubsegmentCheck(S_i)$

1. IF S_i is the subsegment in non-leaf node THEN
2. FOR $k = 1$ to 2 DO //*check the left and right child subsegments of* S_i

3. IF S_{ik} satisfies the condition of Lemma 3 or Lemma 2 THEN
4. Label S_{ik} as **"OS"** or **"NS"**; //S_{ik} *is an Outlier/Non-Outlier Subsegment*
5. ELSE //S_{ik} *is an Undetermined Subsegment,* **check its subsegments or points**
6. $SubsegmentCheck(S_{ik})$;
7. ELSE //S_i *is an Undetermined Subsegment in leaf node,* **check its points one by one**
8. FOR each point p in S_i DO
9. $Count_p = Count_i$;
10. FOR each point $q \in S_j$ where S_j is adjacent to S_i DO
11. IF $ndist(p, q) \leq d$ THEN
12. Increment $Count_p$ by 1;
13. IF $Count_p \leq (1 - P)(N + m_1 - m_2)$ THEN
14. p is an outlier, label S_i as **"US(O)"**; //S_i *has outlier(s)*
15. IF S_i is not labeled as **"US(O)"** THEN
16. Label S_i as **"NS"**;

Note that the only difference between algorithm 2 and algorithm 1 is that in algorithm 2 if S_i is undetermined we make use of the subsegment points count information in binary trees to recursively check subsegments of S_i (by the procedure $SubsegmentCheck(S_i)$) . The points scan does not happen unless the subsegment in leaf node is still undetermined.

5.2 Mining outliers when parameter d or P changes

As we know, the output of the distance-based outliers in a spatial network relies on two parameters d and P. So how to choose the meaningful value of d or P is crucial to the effectiveness of mining results. In practice, users have more experience and better understanding in using percentage P, i.e. the higher value of P means higher degree of the outlierness. On the other hand, since many users are not domain experts, they are not sure which value of d is suitable to the mining algorithm. Instead, they are more likely choosing different values of d for the outlier mining algorithm and evaluate the effectiveness by comparing the outliers with respect to different input parameters. As a result, it is necessary to develop efficient incremental methods for answering outliers in case of updates on d. Since d is always changed, the data structure used in the previous section: a binary tree with multiple level for each edge is not suitable, under such circumstance, we have to just keep one level in each binary tree.

Without loss of generality, we assume each time users input a new $d' = c \cdot u$, where c is a positive integer. Specifically, this corresponds to changing the segments based on the lowest level of subsegments with unit u described in Section 5.1.

Since each segment of length d consists of multiple subsegments where each has length u, thus for any new distance threshold d', it is not necessary to build the new edge-segments from scratch by re-partitioning the edges into segments of the new d' for storage. Instead, we can only maintain the existing edge-segments and each time "virtually" infer the corresponding segments of the new length d' when accessing the existing edge segments for outlier detection. Figure 7 shows

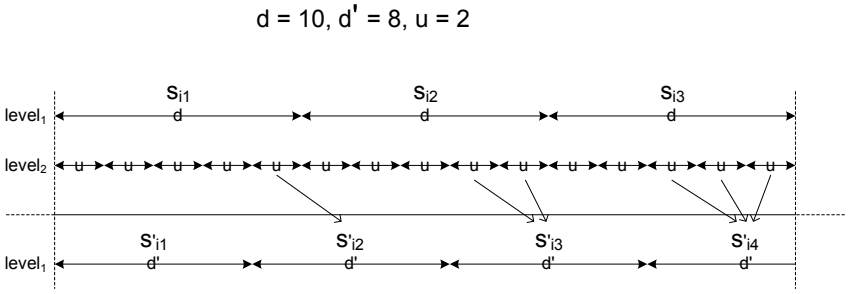

Fig. 7. Segments shifting when d changes to d'

the example that given the edge segments $S_{i1}, S_{i2}, \ldots$ of length d where S_{ij} consists of subsegments with unit length u in leaf nodes, the new edge segments $S'_{i1}, S'_{i2}, \ldots$ of length d' can be inferred by shifting the position of existing subsegment to that of a new subsegment. By updating the counts in the new segment, we can evaluate its new segment type accordingly. Those segments with label **"US"** will be further checked to identify the outlierness of each belonging point. Since the procedure outlier detection is similar to the previous subsection, we omit the detailed description due to the limitation of space. We also have the following interesting property for the inferred new edge segments.

Lemma 4. *Suppose the edge segments of length d' are inferred from edge segments of length d, if $d' > d$ and if segment S'_i of length d' contains any segment S_j of length d labeled as* **"NS"**, *S'_i is labeled as* **"NS"**; *if $d' < d$ and if any segment S'_i of length d' is contained in any segment S_j of length d labeled as* **"OS"**, *S'_i is labeled as* **"OS"**.

For the case of different values of P, there is no need to rebuild or infer the new edge segments, but only updating the type of each segment by applying the similar method in subsection 5.1 to reduce the unnecessary pairwise distance computations. We have the following interesting property for the type of edge segment in the case of new P'.

Lemma 5. *For the segment S_i, if new percentage threshold $P' > P$, then S_i will still be* **"NS"** *if S_i is* **"NS"** *w.r.t. P; if $P' < P$, then S_i will still be* **"OS"** *if S_i is* **"OS"** *w.r.t. P.*

6 Experimental Evaluation

In this section, we evaluate the performance of our proposed techniques. We implemented the naive index based method and the edge-segmentation method for mining outliers. We also implemented the algorithms for mining outliers in dynamic settings with the points being updated and when parameters changing. All algorithms are written in C++ and the experiments were run on a PC

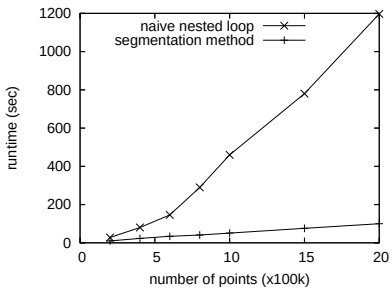

Fig. 8. Static methods

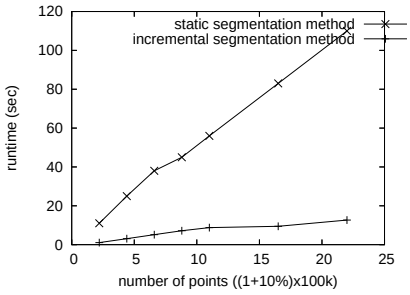

Fig. 9. Dynamic methods(1)

with a Pentium 4 CPU of 1.6GHz, a memory of 512Mb. We used the real road networks of Canada which can be obtained from www.maproom.psu.edu/dcw/ and did some cleaning to form a connected network. In the network, there are 42582 nodes and 46731 edges. The Euclidean distance of the connected nodes is set as the weights of the graph edges, and this is a natural way for the weight setting when we simulate the traffic of the road networks.

On the road networks, we generated points that simulate real world traffic. We start from a random node and use Dijkstra's algorithm to traverse the network and add points to the edges. The way to control the points generated is similar to [24] which focuses on generating the points to form clusters but we expand it to create the outliers. At the border of each cluster, we make the points especially sparse so as to control outlier. By adjusting the magnification factor F and the initial separation distance s_{init}[24], points are generated with different sparsity so that we can test different parameters for the outlier mining. For some fixed d and P, we run the naive nested loop and segmentation based algorithms and they all return the same results. It is also the same when we run our dynamic mining algorithms or compute it from scratch without using the existing results computed earlier. This proves the effectiveness of the algorithms, so we just focus on investigating the efficiency and the scalability of the methods.

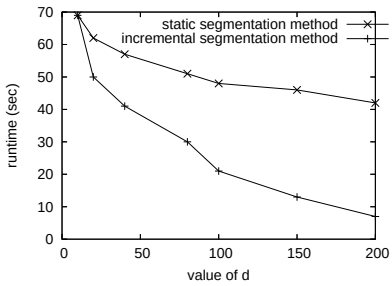

Fig. 10. Dynamic methods(2)

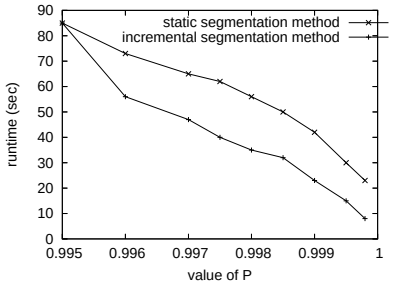

Fig. 11. Dynamic methods(3)

To test efficiency and scalability, we generated sets of points with different cardinality. The cardinality of the points set is from 200k to 2000k where P is a fixed value (0.9985). Figure 8 shows the runtime for these different data sets. We can see that the naive nested loop method is much slower than the segmentation based method which is almost linearly increased with the number of points. If **10%** points is updated on each set of points listed in Figure 9, it shows that our incremental method runs much faster than iterating the static segmentation method on the updated points.

We use a set of 800k points to investigate the performance of proposed methods in the case of changing values of P and d. If d changes, we keep P as 99.5%; while if P changes, d is kept as 20 units(outliers have been computed when $P = 99.5\%$). Both Figure 10 and Figure 11 show that the dynamic segmentation methods perform better than static methods.

Interestingly, we observe in Figure 12 that when P increases, the sum of percentage of **NS** and **US** is almost 100% with respect to the very small percentage of **OS**, which reflects the fact that the larger value of P, the less number of outliers in the spatial network. The similar result can be obtained if we increase the value of d since the larger value of d also leads to less number of outliers.

7 Conclusion

The achieved fruitful results in both research and applications have substantially demonstrated the important role of outlier analysis in data mining area. Existing work on outlier detection involves in different types of databases such as spatial databases, time series databases, bio-medical database etc., while few of them is applied on spatial networks where points reside in every edge. In this paper, we explore the interesting problem of distance-based outlier in spatial networks and propose an efficient mining method which partitions each edge of spatial network into a set of length d segments, then quickly identify the outliers in the remaining edges after pruning those unnecessary edges which cannot contain outliers. We also study the dynamic settings in the spatial network, including updating points or the input parameters of outlier measures are changed. The experimental results verify the scalability and efficiency of our proposed methods.

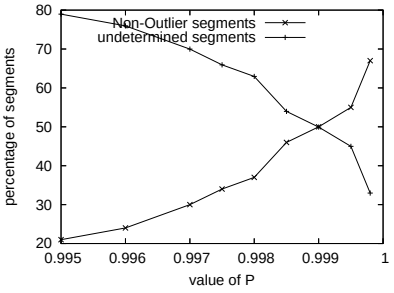

Fig. 12. Dynamic methods(4)

References

1. C. Aggarwal and P. Yu. Outlier detection for high dimensional data. In *SIGMOD* 2001

2. Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, Jorg Sander LOF: Identifying Density-Based Local Outliers. In *SIGMOD* 2000
3. V. Barnett and T. Lewis. In *Outliers in Statistical Data*. John Wiley & Sons, 1994.
4. Deepayan Chakrabarti: AutoPart: Parameter-Free Graph Partitioning and Outlier Detection. In *PKDD* 2004
5. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *KDD* 1996
6. S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *SIGMOD* 1998
7. D. Hawkins. *Identification of Outliers*. Chapman and Hall, London, 1980.
8. V. Hautamki, I. Krkkinen and P. Frnti:Outlier detection using k-nearest neighbour graph, In *ICPR* 2004
9. Jiawei Han, Micheline Kamber: Data Mining: Concepts and Techniques. In *Morgan Kaufmann* Publishers.
10. H. Jagadish, N. Koudas, and S. Muthukrishnan:Mining deviants in a time series database. In *Proc. VLDB* 1999
11. W. Jin, Anthony.K.H. Tung and J.W. Han. Mining Top-n Local Outliers in Large Databases. In *KDD* 2001
12. Edwin M. Knorr, Raymond T. Ng: Algorithms for Mining Distance-Based Outliers in Large Datasets. In *VLDB* 1998
13. E. Knorr and R. Ng. Finding Intensional Knowledge of Distance-Based Outliers. In *VLDB* 1999
14. S. Muthukrishnan, Rahul Shah, Jeffrey Scott Vitter: Mining Deviants in Time Series Data Streams. In *SSDBM* 2004
15. R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *VLDB* 1994
16. Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, Christos Faloutsos. LOCI: Fast Outlier Detection Using the Local Correlation Integral. In *ICDE* 2003
17. Spiros Papadimitriou, Christos Faloutsos: Cross-Outlier Detection. In *SSTD* 2003
18. N. Roussopoulos, S. Kelley and F. Vincent. Nearest neighbor queries,*SIGMOD* 1995
19. Sridhar Ramaswamy, Rajeev Rastogi, Kyuseok Shim: Efficient Algorithms for Mining Outliers from Large Data Sets. In *SIGMOD* 2000
20. Shashi Shekhar, Chang-Tien Lu, Pusheng Zhang: Detecting graph-based spatial outliers: algorithms and applications (a summary of results). *KDD* 2001
21. Jorg Sander, Raymond T. Ng, Monica C. Sleumer, Man Saint Yuen, Steven J. Jones: A methodology for analyzing SAGE libraries for cancer profiling. In *ACM Trans. Inf. Syst.* 23(1): 35-60 (2005)
22. Weng-Keen Wong, Andrew W. Moore, Gregory F. Cooper, Michael Wagner: Rule-Based Anomaly Pattern Detection for Detecting Disease Outbreaks. *AAAI* 2002
23. Man Lung Yiu, Nikos Mamoulis: Clustering Objects on a Spatial Network. In *SIGMOD* 2004
24. Man Lung Yiu, Nikos Mamoulis, Dimitris Papadias: Aggregate Nearest Neighbor Queries in Road Networks. In *IEEE Trans. Knowl. Data Eng.* 17(6): 820-833 (2005)
25. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD* 1996