ABSTRACT

The detection of outliers in spatio-temporal traffic data is an important research problem in the data mining and knowledge discovery community. However, to the best of our knowledge, the discovery of relationships, especially causal interactions, among detected traffic outliers has not been investigated before. In this paper we propose algorithms which construct outlier causality trees based on temporal and spatial properties of detected outliers. Frequent substructures of these causality trees reveal not only recurring interactions among spatio-temporal outliers, but potential flaws in the design of existing traffic networks. The effectiveness and strength of our algorithms are validated by experiments on a very large volume of real taxi trajectories in an urban road network.

Categories and Subject Descriptors
H.2.8 [Database Applications]: Data mining

General Terms
Algorithms

Keywords
Spatio-temporal outliers; outlier causalities; frequent substructures; urban computing and planning;

1. INTRODUCTION

The increasing availability of location-acquisition technologies including GPS and WIFI have resulted in huge volumes of spatio-temporal data, especially in the form of trajectories [2, 5, 7, 15, 14, 19, 17]. Unusual patterns of moving objects’ trajectories generally reflect abnormal traffic streams on road networks, which could be caused by aperiodic events including celebrations, parades, large-scale business promotions, protests, traffic control and traffic jams. Therefore, the detection of outliers/anomalies from trajectory data can help in sensing abnormal events and plan for their impact on the smooth flow of traffic. In this study, we treat both known (planned) and unknown (unplanned) events that behave differently from daily network traffics as anomalies.

Challenges and Contributions

In order to successfully detect outliers and causal interactions among them, the following challenges need to be addressed: (i) Heterogeneous traffic patterns: the traffic patterns on roads vary across days of a week and hours of a day. Different road segments have often distinct time-variant traffic patterns. It is difficult to use one model to detect outliers across the road network at different time periods. (ii) Data sparseness and distribution skewness: even though we could have a large number of sensors (taxis) probing the traffic on roads, there are many roads that have only a small number of samples given a large size of road networks in a major city. Moreover, a few road segments are traveled by thousands of vehicles in a few hours, while some segments may be only driven on a few times in a day. These two properties together result in unique challenges in detecting outliers from traffic data. (iii) Causality among outliers: we not only need to discover outliers from the traffic, but also infer causal relationships and interactions among them, especially given the large number of outliers that could be identified. So a challenge is how to detect the appearance, growth, disappearance and transformation of outliers by time (e.g., propagation of a traffic jam).

In this paper we design several steps to address the above challenges and propose solutions to the problem of detecting spatio-temporal outliers and causal relationships among them from traffic data streams. We use contexts of road networks in this study, however, algorithms proposed in this paper can be generally applied to domains of networking [12,
The problem of outlier monitoring has also been studied in [2] which builds local clusters on trajectory streams and monitors outliers that are defined by a “trajectory” (instead of a spatial link as ours). Another method is Sun et al. [22] where a measure, spatial local outlier measure (SLOM), is proposed to capture the local behavior of datum in their spatial neighborhood. This measure takes into account the local stability around a data point and suppresses the reporting of outliers in highly unstable areas. A generalized local statistical (GLS) framework is proposed in [6] which studies the performance of local based methods on detecting outliers in geo-statistical data with either linear or nonlinear trends, and compares them against global based methods. Wu et al. [23] design algorithms detecting the most abnormal discrepancy regions in precipitation data, where they use four sweep lines to form grids which are treated as regions. However, none of these approaches model and capture temporal relations (causalities) among detected outlying regions. Lee et al. [15, 14] have designed a “partition-and-group” framework for clustering and detecting trajectory outliers. In their approach, they first partition the trajectories into small segments and then use both distance and density to detect abnormal sub-trajectories. This is different from our work as we detect abnormal regions and links of the entire traffic network (instead of objects moving in the network).

Moving objects are usually associated with periodic behavioral patterns, and there have been several methods proposed to address the problem of detecting such periodic movements [3, 4, 10, 9, 16]. Cao et al. [3, 4] proposed abbreviated list tables (ALT) to find subsequences that appear periodically and frequently in data sequences, but the periodic patterns they detect are very sensitive to parameter settings. Similarly, Elkeky et al. [9] have proposed specific definitions of periodicities and algorithms for identifying the periodic patterns.

3. OVERVIEW

In this section, we introduce our notations, definitions and the main structure of the proposed model.

3.1 Preliminary Concepts

The overall traffic map is partitioned into regions (Rgn) bounded by high level (i.e. major) roads, each of which may consist of a number of road segments. Figure 1(a) and 1(b) demonstrate an example of region formations.

**Definition 1.** Trajectory: A trajectory $Tr$ is a trace cre-
at ed by a moving object in geographical space. A \( T_r \) is represented by a set of time-ordered points, e.g. \( T_r : p_1 \rightarrow p_2 \rightarrow ... \rightarrow p_n \), where each point consists of a geospatial coordinate set and a timestamp, i.e. \( p = (\text{longitude}, \text{latitude}, \text{timestamp}) \).

**Definition 2.** Transition: Given a trajectory \( T_r : p_1 \rightarrow p_2 \rightarrow ... \rightarrow p_n \), there exists a transition \( S \) between \( Rgn_1 \) and \( Rgn_2 \), if there exists adjacent points \( p_i \) and \( p_{i+1} \) (\( 1 \leq i \leq n + 1 \)) such that \( p_i \) is in \( Rgn_1 \) and \( p_{i+1} \) is in \( Rgn_2 \), and \( Rgn_1 \) is not the same to \( Rgn_2 \). A transition is associated with a leaving time (timestamp of \( p_i \)) and an arriving time (timestamp of \( p_{i+1} \)).

**Definition 3.** Link: A link (Lnk) is comprised of a pair of regions (\(<Rgn^a, Rgn^b>\)) indicating a virtual spatial connection between the origin region and the destination region. There exists a link from one region (\( Rgn^a \)) to another (\( Rgn^b \)) if and only if there at least one object moving from \( Rgn^a \) at timestamp \( t_s \) to \( Rgn^b \) at \( t_{s+1} \). Figure 1(c) and 1(d) give examples of links.

**Definition 4.** Time frame: A time frame (\( t_f \)) is a set of consecutive time intervals\(^1\). Figure 3(a) shows an example of a time frame.

**Definition 5.** Spatio-temporal outlier: A spatio-temporal outlier (STO) is a link whose non-spatial and non-temporal attribute values are very different from the values of its spatio-temporal neighbors. Spatio-temporal neighbors of a link \( Lnk_i \) are the links whose locations and timestamps are close to those of \( Lnk_i \).

**Definition 6.** Outlier causality: STO_2 (with a region pair \(<Rgn^a, Rgn^b>\) and a time frame \( t_f \)) is caused by STO_1 (containing a region pair \(<Rgn_1^a, Rgn_1^b>\) and a time frame \( t_{f1} \)) if and only if the following conditions hold true: (i) The destination of STO_1 is the same as the origin of \( STO_2 \) (i.e. \( Rgn_1^a = Rgn_2^b \)); (ii) Time frames \( t_{f1} \) and \( t_f \) are consecutive to each other and \( t_f \) is ahead of \( t_{f1} \).

During the construction of an outlier tree, an outlier \( STO_i \) is a child of another outlier \( STO_j \) if \( STO_i \) is caused by \( STO_j \).

3.2 Framework

The main structure of our model is illustrated in Figure 2. The three main steps are preprocessing traffic data to build a region graph, detecting outliers and finally discover causal relationships between the discovered outliers. The second and the third step have three and two sub-steps respectively. Details of these (sub-)steps are described in the following section.

4. METHODOLOGY

In this section, we provide details of our model as shown in Figure 2. Specifically, we focus on the detection of spatio-temporal outliers based on each link’s “distort”, construct outlier causality trees based on these outliers, and discover the most frequent causal trees which are indicative of recurrent traffic abnormalities.

4.1 Building the Graph of Regions

In our study, we assume the map of traffic network, the set of major roads, and the trajectories of objects are all known. Although it is more straightforward to apply a simple “\( n \times m \)” grid on maps to define regions, cells of a grid are equal-sized and do not reflect natural differences of regions in a traffic network. So instead of using equal-sized grids, we define regions of a traffic network by road segments as illustrated in Figure 1. In detail, we build a graph of regions according to the following three steps.

1. **Region Partitioning:** As shown in Figure 1(a) and Figure 1(b), we partition a city into dis-jointed regions using the major roads of the city. Here we employ Connected Components Labeling (an image segment method) [21] to partition a map into regions effectively and efficiently, since the problem of subdivisions in a polygonal region is known to be NP-complete [11].

2. **Formulating transitions:** By scanning the trajectory data set, we transfer each trajectory into a sequence of transitions between pairs of regions in terms of definition 2. As demonstrated in Figure 1(c), a trajectory passing three regions \( a, b, \) and \( c \) results in two transitions: \( a \Rightarrow b, \) and \( b \Rightarrow c \).

3. **Generating links:** If there is a transition generated between two regions, we connect the two regions with a link (refer to Definition 3). In timebin \( j \), a link \( i \) (\( Lnk_i = <Rgn^a, Rgn^b> \)) is associated with a feature vector of three properties \( f_{i,j} = <\#Obj, Pct_o, Pct_a> \):

(a) \#Obj: Total number of objects on the links (i.e. objects moving from \( Rgn^a \) to \( Rgn^b \) in this timebin);

(b) Pct_o: The proportion of \#Obj among all objects moving out of \( Rgn^a \) in this timebin;

(c) Pct_a: The proportion of \#Obj among all objects moving into \( Rgn^b \) in this timebin;

Then, using Figure 1(d) as an example (where the number shown on each link is the number of transitions pertaining to the link), the property of link \( a \Rightarrow b \) is \( f_{i,j} = <\#Obj=2, Pct_o=0.4, Pct_a=0.25> \).

4.2 Detecting Outliers from Graph Links

Assume each time frame is comprised of a fixed number of \( q \) timebins. Given a time frame \( t_{f1} \), we denote the sequence of feature values of a link \( Lnk_i \) in this time frame by:

\[ F_{i,j} = <f_{i,j,q+1}, f_{i,j,q+2}, ..., f_{i,j}> \] (1)
For each link ($L_{nk}$) in each time frame $tf_i$, we calculate the distortion between two time frames (denoted by $minDistort_{i,j}$) by searching for the minimum difference between $tf_j$ and the same time frames of the same days on consecutive weeks.

With this approach $minDistort$ is capable of capturing the special pattern of traffic data that similar behaviors are observed during the same time of different days or the same day of different weeks etc.

Algorithm 1 shows the procedure of calculating distorts. In line 7 of the algorithm, we obtain the difference between two time frames of a link by computing their Euclidean distance:

$$Distance(tf_j, tf_i, L_{nk}) = \sqrt{\sum_{k=0}^{g-1}||f_{i,j,k} - f_{i,t,k}||^2}$$  \hspace{1cm} (2)

We use $minDistort_{i,j}$ obtained from Algorithm 1 as the “non-spatial and non-temporal attributes” (see Definition 5) of each link in each time frame. Extreme values among $minDistort$ of all links are identified as temporal outliers. By subtracting the $min$ and dividing by the $max$ the feature values of the links are in the range of $[0,1]$. The normalization removes the effect of different regions and volume sizes. Another advantage of using $minDistort$ is that it prevents the examination of many repeating patterns (where $minDistort \approx 0$).

Then for each time frame, there is a corresponding three dimensional vector (formed by features <#Obj, Pct_o, Pct_d>) shown in Figure 4. As each point represents a link, we identify the most extreme points as outliers in that time frame. To normalize the effect of variances along different directions, we use the Mahalanobis distance (instead of Euclidean distance) to measure the the extremeness of data points. We use the Mahalanobis distance here in order to normalize the distance by the variance in different directions.

In this way, the outliers detected are links whose features have the largest difference from both their temporal neighbors (for using “$minDistort$”) and spatial neighbors (for being detected “among all links”) – so they are spatio-temporal outliers (STOs). Another advantage of identifying “extreme points” as outliers is that it can detect abnormal links with either too low volumes or too high volumes since extremeness of points are based on their Mahalanobis distances.

Now each STO is a spatial link associated with a time frame. We represent a STO by its link $L_{nk}$ (containing an original region and a destination region) and its time frame $tf_j$, i.e., $STO_{i,j} = <Rgn_o, Rgn_d, tf_j>$.

### 4.3 Constructing Outlier Trees

We propose an algorithm named $STOTree$ that finds outlier dependencies by looking at the relationship of outliers from the earliest time frame through the last. The main insight of $STOTree$ is that an outlier $STO_1$ is a parent of another outlier $STO_2$ if $STO_1$ occurred before
Algorithm 2 STOTree: constructing all outlier trees

Input: STOutlier: a set of spatial-temporal outliers of size $t \times k$ where $t$ is the number of time frames, and $k$ is the number of outliers to examine in a time frame.

Output: STOTrees: a list of roots of spatial-temporal trees.

1: $STOTrees \leftarrow \emptyset$;
2: for Each time frame $i$ ($i \in \{1, ..., t\}$) do
3:   for Each outlier $j$ ($j \in \{1, ..., k\}$) in time frame $i$ do
4:      $STORoot_{i,j} \leftarrow \text{FindAllChildren}(STOutlier_{i,j,1});$
5:      $STOTrees \leftarrow STOTrees \cup STORoot_{i,j};$
6:   end for
7: end for
8: Return $STOTrees$;

Subroutine: FindAllChildren($STOutlier_{i,j,1}$, $i$)
9: if Time frame $i$ is the last time frame then
10:   Return $STOutlier_{i,j};$
11: end if
12: $STOutlier_{i,j,1}\text{.subnodes} \leftarrow \emptyset$
13: for Each outlier $u$ ($u \in \{1, ..., k\}$) in time frame $i + 1$ do
14:   if $STOTrees$ contains $STOutlier_{i+1,u} \text{.subnodes}$ then
15:       continue;
16:   end if
17: if $STOutlier_{i,j,1}\text{.Rgn}^{u} == STOutlier_{i+1,u}\text{.Rgn}^{u}$ then
18:   $STOutlier_{i,j,1}\text{.subnodes} \leftarrow STOutlier_{i,j,1}\text{.subnodes} \cup \text{FindAllChildren}(STOutlier_{i+1,u}, i + 1);$
19: end if
20: end for
21: Return $STOutlier_{i,j};$

STOT in time and they are spatially correlated. Algorithm 2 demonstrates the process of constructing outlier trees from discovered outliers. Note the algorithm results in a collection of trees (a forest). The subroutine (Line 9 to 21) is a recursive function used to retrieve all possible descendants of a node. For each time frame, this recursive function is called on each outlier of the current time frame to compare with each outlier of next time frame, unless the “current” outlier tree already contains outliers of next time frame (Line 14 to 16). So the overall time complexity of the outlier tree construction process on each time frame is upper bounded by $O(k^2)$, where $k$ is the number of outliers in a time frame.

We do not place a restriction on the maximum size of outlier trees in the STOTree algorithm, under the assumption that abnormal events caused by one single accident are not expected to last for a long time and the size outlier trees should not grow infinitely. In Section 5.3 we provide empirical evidence that confirms the maximum size of trees is usually small.

Now we give an example by using Figure 5 to demonstrate the process of Algorithm 2 for building outlier trees. Figure 5 uses top 3 outliers in three consecutive time frames, so the input parameters in Algorithm 2 in this case are $k = 3$ and $t = 3$. The algorithm starts from time frame 1 (Line 2 of Algorithm 2), and for each of the top three outlying links (Line 3 to 6), i.e., $A \Rightarrow B$, $C \Rightarrow D$ and $E \Rightarrow F$, the algorithm searches in time frame 2 (Line 13 to 20) and checks whether there is any following link that can be a child of a previous link (Line 17 to 19). This allows the algorithm to find outlying links $B \Rightarrow G$ and $B \Rightarrow E$ as children of $A \Rightarrow B$; and similarly it identifies link $H \Rightarrow K$ in time frame 3 as a child of $J \Rightarrow H$ in time frame 2. Therefore two outlier trees are built up as shown in the right side of Figure 5. In this way, Algorithm 2 scans through all time frames of traffic data we have, and builds a forest of various outlier trees.

4.4 Causal Outlier Detection

Denote by $T$ the forest containing all outlier trees. The most significant and recurring causal relationships correspond to the most frequent subtrees of $T$. The mechanism of discovering frequent subtrees from all outlier trees is inspired by the process of mining frequent item sets, except that we design our own strategy to generate frequent subtree candidates (through node insertion on trees).

The process of discovering frequent subtrees from constructed outlier trees is shown in Algorithm 3. Given a predefined support threshold $\epsilon$, we first find all single nodes whose supports exceed $\epsilon$ (Line 3 of the algorithm), then we use this set of frequent single nodes to form candidates of frequent subtrees. The “while” iteration (Line 6 to 30) first generates candidates of subtrees (Line 9 to 15), and then checks the support of each candidate and performs filtering (Line 18 to 29) according to $\epsilon$.

When generating subtree candidates, new subtrees (whose sizes are increased by one) are created by inserting a frequent single node into previous frequent subtrees. This node insertion process is given in Algorithm 4. The single node to be inserted is first compared with the root of the tree, and is inserted as a subnode of the root (Line 1 to 3 of Algorithm 4) if the root can be a parent of the single node and its existing children do not contain the single node. Otherwise, the single node is compared and checked whether it can be inserted into branches below the root (i.e. a recursive process shown in Line 8 to 12). When counting the support of a candidate subtree, we increase the frequency of the candidate by one if all nodes (with their immediate subnodes) of the candidate have an exact match with a discovered outlier tree (Line 21 to 23).

The effectiveness and strengths of our algorithms, STOTree and frequentSubtree, are evaluated in the next section.

5. EXPERIMENTS AND ANALYSIS

In this section we report on the experiments carried out on taxi trajectory data on the road network of Beijing city.
Algorithm 3 frequentSubtree: discovering frequent subtrees from STOutlier trees

Input: STOTrees: a list of roots of spatial-temporal trees; ε: a support threshold for frequent substructure selection.
Output: frequentSubtrees: a list of roots of frequent spatial-temporal subtrees.

1: // Form a list of frequent nodes (i.e. frequent trees of size 1).
2: numTrees ← number of roots in STOTrees;
3: frequentNodes ← unique nodes appearing at least numTrees × ε times in STOTrees;
4: mergeTarget ← frequentNodes;
5: frequentSubtrees ← an empty set();
6: while size(mergeTarget) > 0 do
7:   // Form candidates of frequent subtrees;
8:   subtreeCandidates ← an empty set();
9:   for Each node singleton, in mergeTarget do
10:      for Each root rooti in mergeTarget do
11:         if nodelnesion(rootj, singleton) then
12:            subtreeCandidates ← subtreeCandidates ∪ rootj;
13:            end if
14:      end for
15:   // Filter subtree candidates by threshold of support ε;
16:   Clear mergeTarget;
17:   for Each candidate candidatei, in subtreeCandidates do
18:      count ← 0;
19:      for Each root rootk in mergeTarget do
20:         if rootk contains candidatei then
21:            count ← count + 1;
22:         end if
23:      end for
24:      if count > ε × numTrees then
25:         frequentSubtrees ← frequentTrees ∪ candidates;
26:         mergeTarget ← mergeTarget ∪ candidates;
27:      end if
28:   end if
29: end for
30: end while
31: Return frequentSubtrees;

Table 1: Statistics of regions and links in the graph built from road network traffic.

<table>
<thead>
<tr>
<th>#Regions</th>
<th>#Links</th>
<th>Avg. #Obj</th>
<th>Avg. Pct_o</th>
<th>Avg. Pct_d</th>
</tr>
</thead>
<tbody>
<tr>
<td>396</td>
<td>10109</td>
<td>742.48</td>
<td>27.07</td>
<td>27.7702</td>
</tr>
</tbody>
</table>

Our experiments are conducted on a 64 bit server with 3.2 GHz CPU and 8 GB memory. We note that although we are using road traffic data, our methods and algorithms can be easily adapted into other domains such as finding anomalies in the internet traffic data and even climate data.

5.1 Data and Parameters

Data: We test our algorithms based on a real GPS trajectory dataset generated by 33,000 taxis in a period of 6 months (from 01/03/2009 to 31/08/2009) [25, 24]. The total distance traveled by the taxis is more than 800 million kilometers and the total number of GPS points is nearly 1.5 billion. The average sampling interval and average distance of between two consecutive points are around 3.1 minutes and 300 meters respectively.

We define ten minutes as a timebin, and six timebins as a time frame (hence one time frame represents an hour). We check two adjacent weeks in minDistort calculations (i.e. the parameter t in Algorithm 1).

Road Network: We perform evaluations based on the road network of Beijing, which contains 106,579 road nodes and 141,380 road segments.

5.2 Results from Building Graphs

The statistics (e.g. number of regions and links) of the graph built are presented in Table 1. We note that in our evaluations, the number of links built by using 1 month, 2 months, ..., until all 6 months of taxi trajectories remains constant at 10109. This observation implies that although the links are dynamic\(^2\) (dependent of traffic data), the model we have built does not miss any links even if one uses only a small fraction of taxi trajectories to build the underlying traffic graph. The distribution of the three features of links is presented in Figure 6.

5.3 Evaluations on Outlier Trees

In this experiment, we bound the value of k (i.e. number of outliers to be identified in each time frame) between 1 and 99, and report its effects on constructing outlier trees (shown as in Figure 7 and 8).

We set the minimum size of a tree (i.e. total number of nodes) to 2, and hence ignore singles nodes (trees of size 1) in counting final outlier trees. What we observe from Figure 7 is that, although the total number of trees constructed by detected outliers increase substantially when k increases (left subfigure), the maximum size of all trees has

Algorithm 4 nodeInsertion: inserting a node to an outlier tree.

Input: Root: a root of an outlier tree; Singleton: a node to be inserted.
Output: true/false: whether or not the node insertion is successful.

1: if Root.Rgn_i equals singleton.Rgn_o & Root.subnodes does not contain singleton then
2:   Root.subnodes ← Root.subnodes ∪ singleton;
3:   Return true;
4: else
5:   if size(Root.subnodes) == 0 then
6:     Return false;
7:   else
8:      for Root of each subnode subRoot in Root.subnodes do
9:         if InsertNode(subRoot, Singleton) then
10:            Return true;
11:       end if
12:      end for
13:   end if
14: end if
15: Return false;

Figure 6: Histograms of the three features (i.e. <#Obj, Pct_o, Pct_d>) of all links in the region graph we obtained.

\(^2\)By contrast, regions are always static (independent of traffic data).
In Section 4.3 we have explained that the time complexity of our outlier tree construction algorithm STOTree is in the worst case (i.e. upper bounded by) $O(k^2)$. Here we present empirical results that illustrate STOTree runs much faster than $O(k^2)$ in practice. From Figure 8 we can observe that the average time used for building trees in a time frame increases almost linearly (instead of quadratically) with $k$. This indicates STOTree can potentially be used in an online setting, and can detect outlier causalities on the fly.

Moreover, the spatio-temporal outliers and their causal interactions detected by our algorithms all coincide with known abnormal events. The following are two prominent examples of known events:

**Known event 1.** The Beijing–HongKong highway was under traffic control for building viaducts of a Beijing light rail (i.e. the light rail of Fangshan line\(^3\)) at midnight of the May 5th, 2009.

**Known event 2.** The entry fee into Olympic sports center was waived during day time on 7th August, 2009.

By using our model, the above two known outliers are successfully detected as shown in Figure 9(a) and 9(b) respectively. However, they are not detected by PCA based techniques as in [13]. As Figure 10 shows, the links of these two known abnormal events are among other points and are undistinguishable in the coordinate framework formed by the first two principle components from PCA.

A major reason for the failure of PCA is that the first known outlier (event 1) occurred in the off-peak hour of most links, and the second known outlier (event 2) happened in the peak hour of most links – in other words, they are captured by the smallest principle components (PCs) and the largest PCs respectively. The difficulty of choosing appropriate PCs makes it hard to use PCA to detect both of these two traffic events.

5.4 Evaluations on Frequent Subtrees

In this experiment, we control the value of support threshold $\epsilon$, and report its effects on discovering frequent outlier subtrees in terms of the number of subtrees passing through $\epsilon$ and their maximum support. We test the properties of frequent subtrees on five sets of support threshold: $\epsilon \in \{0.01, 0.05, 0.1, 0.15, 0.2\}$. We ignore values of $\epsilon$ higher than 0.2, since the total number of frequent subtrees is already considerably small when $\epsilon = 0.2$ (as shown in Figure 11(a)). However, as long as the number of subtrees is higher than zero, the frequentSubtree algorithm can still identify the most frequent subtree that has the highest support (as shown in Figure 11(b) on $\epsilon = 0.15$). This observation illustrates the completeness of the frequent subtrees generated by the frequentSubtree algorithm.

Recall that regions indicated by the most frequent (i.e. $\epsilon$...
with highest support) subtrees are the ones that have strategi-

cal design drawbacks from the perspective of urban road

network planning. For example, when \( k = 7 \), the top two fre-

quent subtrees fall in the suburb of “Wangjing” and “Laoshan”

respectively, shown in Figure 12. These subtrees indicate

that both of the two suburbs are much more frequently

overloaded with vehicles than other suburbs, and are in

need of public transportation systems (e.g. subways) pass-

ing through them to reduce the need of commutations on

ground. Such indications coincide with the future subway

construction plan\(^4\) of Beijing city: (i) New subway lines of

Line 14 and Line 15 will be launched to travel through the

suburb of “Wangjing” in year 2011 and 2013 respectively; (ii)

Subway Line 10 (second construction stage) to be put into

use by year 2012 will be centered at the suburb of “Laoshan”.

These relationships between the official subway construc-

tion plans and our frequent subtrees validate the correctness

of the \textsc{frequentSubtree} algorithm, and demonstrate the ca-

http://en.wikipedia.org/wiki/Beijing_Subway

Figure 12: Top two frequent subtrees and suburbs they cover (“Wangjing” and “Laoshan”) when \( k = 7 \). These two subtrees suggest that there are potential design flaws in the current road networks spanning the two suburbs. These results coincide with (and hence are validated by) future construction plan of Beijing subways.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have studied the problem of detecting spatio-temporal outlier and their causal interactions from traffic data streams. We have proposed \textsc{STOTree}, an algo-

rithm for discovering spatio-temporal outliers and causal re-

lationships between them. While the worst-case time com-

plexity of \textsc{STOTree} is quadratic (in the number of outliers in
each time frame), empirical evidence strongly suggests that the complexity is closer to linear time.

We have also proposed a \textsc{frequentSubtree} algorithm which can be used to reveal recurrent anomalies in the road net-

work. Based on the \textsc{STOTree} and \textsc{frequentSubtree} algo-

rithm we were able to identify real and valid instances of anomalies in Beijing traffic data. This suggests that our

approach has the potential of contributing to a new data driven approach towards road traffic analysis.

In future our plan is to apply and extend the use of our algorithms in the domain of internet traffics etc.
7. REFERENCES


