Discovering Congestion Propagation Patterns in Spatio-Temporal Traffic Data

Hoang Nguyen
National ICT Australia
13 Garden St
Eveleigh, NSW 2015
hoang.nguyen@nicta.com.au

Wei Liu
University of Technology Sydney
Ultimo, NSW 2015
wei.liu@uts.edu.au

Fang Chen
National ICT Australia
13 Garden St
Eveleigh, NSW 2015
fang.chen@nicta.com.au

ABSTRACT

Congestion is the condition of the road in the traffic networks which is characterised as slow speed and long travel time. The detection of unusual traffic patterns including congestions is an significant research problem in the data mining and knowledge discovery community. However, to the best of our knowledge, the discovery of propagation, or causal interactions among detected traffic congestions has not been appropriately investigated before. In this research, we introduce algorithms which construct causality trees based on temporal and spatial information of identified congestions. Frequent substructures of these causality trees reveal not only recurring interactions among spatio-temporal congestions, but potential bottlenecks or flaws in the design of existing traffic networks. Our algorithms are validated by experiments on a large real-time travel time data in an urban road network.

Keywords

Congestion propagation, spatio-temporal, causal, frequent substructures, urban computing and planning

1. INTRODUCTION

Traffic congestion is considered as one of the most important issues in many cities over the world. Congestions usually happen during peak hours or were caused by aperiodic events including celebrations, parades, large-scale business promotions, protests, traffic controls and incidents. When congestions occurred in one area of the urban traffic network, they are likely to affect the traffic flows of surrounding areas, especially to all the traffic leading to the congested road. Hence, it is essential to develop an efficient methods to discover frequent patterns of congestion propagations in the traffic networks. Traffic management systems can potentially benefit from this in preventing and clearing traffic congestion in time or making appropriate proposal for future development of the traffic networks.

Recent research in traffic networks is aided by the increasing availability of location-acquisition technologies including GPS and WIFI with vast volumes of spatio-temporal data, especially in the form of trajectories [4, 3, 2, 14, 13, 15, 27]. However, in order to successfully detect congestions 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 congestions across the road network at different time periods. (ii) Data sparseness and distribution skewness: even though a large number of sensors probing the traffic on roads are available, 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 processing traffic data. (iii) Causality among congested segments: given the large number of congested segments that could be identified, the challenge is how to detect the appearance, growth, disappearance and transformation of congestions by time (e.g., propagation of a congestion).

The method introduced in this paper provides solutions to the above problems of detecting spatio-temporal congested sites and causal relationships among them from traffic data streams. The context of road networks was utilised in this study, however, the algorithms proposed in this paper can be generally applied to spatial temporal domains, such as networking and climate change etc [9, 19]. More specifically, the system comprises of three components:

1. Congestion tree construction: we propose a STCTree algorithm based on both spatial and temporal properties of detected congestions (which are certain "road segments" in a snapshot) to construct congestions trees, which uncovers causal relationships among the congested segments.

2. Frequent congestion subtree discovery: we propose a frequent Subtree algorithm, inspired by association rule mining, which generates the most frequent sub-structure (subtree) from all discovered congestion trees. These frequent subtrees reveal recurrent congestion trees in the data and suggest inherent problems in existing road networks.

3. Traffic congestion propagation modelling and causality

probability estimation using Dynamic Bayesian Network.

2. RELATED WORK

Instead of investigating the causal interaction between congested roads which are dynamic in term of time and space, there have been a number of efforts on analyses of flow characteristic at bottlenecks in the networks [22, 24, 10, 23, 8]. In general, the characteristic of real-time traffic flow propagation is extremely complicated to be modeled because it relates to human behaviour [18]. Thus, when conducting research on the formation of congestions, the traffic networks were usually simulated using the cell transmission model and average journey velocity [6, 7]. Then, different traffic demand conditions could be employed to investigate the capability of links and identify the distribution of bottlenecks in the network [18].

To the best of our knowledge, the only research that proposed the problem of discovering casual relationships among spatio-temporal congestions was introduced by Liu et al [17, 25, 5]. However, there are several problems in previous methods:

1. The traffic networks were modelled by partitioning the urban area into regions [17, 25] or junctions [5]. However, the real traffic flows are road-based not region-based. For example, there are several routes from one region to nearby region and each route can have different traffic conditions or when a congestion happened at a specific road, it is hard to be modelled by looking at the whole region status.

2. The recursive implementation of the causal tree algorithm is usually not applicable for large traffic network given limited time and memory.

3. The completeness of the frequent subtree algorithm is subject to the order of the frequent nodes during tree construction process.

In the present work, we have designed several steps to address the above limitations. More specifically, the improvements and contributions we make in this paper are:

1. Model the traffic network based on the main roads in the urban area where sites are sensor points and segments are the road between two connected sensors.

2. Propose dynamic programming approach to build spatial-temporal trees which shows the better solution in term of time-efficiency and memory usage than recursive algorithm.

3. Propose the correctness and completeness frequent subtrees algorithm which extends the Apriori approach in mining frequent item sets.

4. Propose general method for congestion propagation modelling and estimation using Dynamic Bayesian Network.

A Bayesian network (BN) is a probabilistic graphical model which comprises of a set of random variables and their conditional dependencies via a directed acyclic graph [26]. A dynamic BN (DBN) is a Bayesian network that models sequence of variables such as a time series or stochastic process [11]. The term "dynamic" means we are modelling a temporal system, not the network's structure changes over time. The simple instance of DBN is the Hidden Markov Model [12].

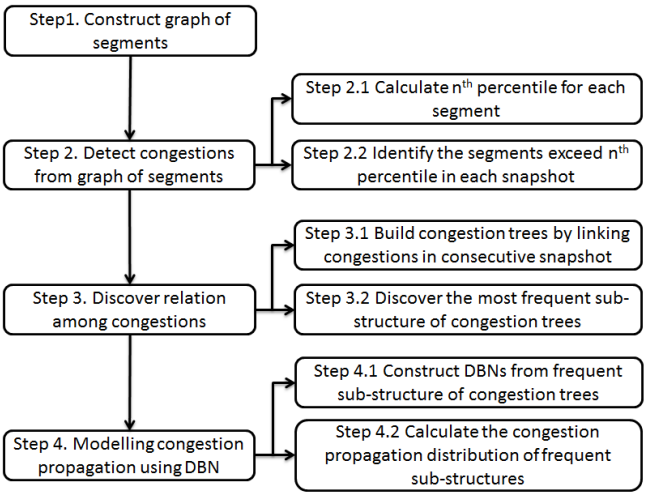


Figure 1: The processing pipeline for out model of detecting spatio-temporal congestions and their inter-causalities.

3. OVERVIEW

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

3.1 Definitions

The overall traffic map comprises of road segments where travel times of the vehicles are recorded at every fixed time interval.

Definition 1. Site: is the location (latitude and longitude) of the sensor on the road.

Definition 2. Segment: A segment (seg) represents the connected sites. A segment contains source site and destination site. The travel time will be recorded for each segment of the network.

Definition 3. Link: A link (Lnk) is comprised of a pair of segments (Seg_s, Seg_d) indicating a virtual spatial connection between the source segment and the destination segment. There exists a link from one Seg_s to another Seg_d if the destination site of the Seg_s is the same as source site of the Seg_d.

Definition 4. Snapshot: is a storage of the travel time from all segments in the network at a given time. A snapshot is identified by the time when all the sensors start to record the travel time of the vehicles within the observed segments.

3.2 Processing Pipeline

The main processing pipeline of our model is illustrated in Figure 1. The three main steps are processing traffic data to build the graph of segments, detecting congestions and finally discover causal relationships between the identified congestions. The following section presents the details of the component steps in the pipeline.

4. CAUSAL TREE DISCOVERY

4.1 Constructing the Graph of Segments

In our study, the map of traffic network and the set of major roads are constant. Because the sites are located at fixed locations, we first construct the static spatial network from the data stream received. To reflect the nature flow of the traffic stream, each segment of the network can be one-way or two-way corresponding to the characteristic of the contained road. In case the segment is two-way, each direction is processed independently because the traffic conditions can be very different between two directions of the same road. Hence, the sparse adjacency matrix is employed to represent the spatial traffic network.

4.2 Detecting Congestions

At the specific snapshot, the travel time between segments are not comparable because each segment has different characteristics such as length, number of lanes and speed limits. To identify the congestions, the evaluation of real travel time was done by different percentiles of *travel time range* from each segment. The i^{th} percentile is the value below which i percent of the observations may be found. In other words, a segment is considered as congested at a specific snapshot if its average travel time is longer than i^{th} (i range from 50 to 95) percentile of overall travel time distribution.

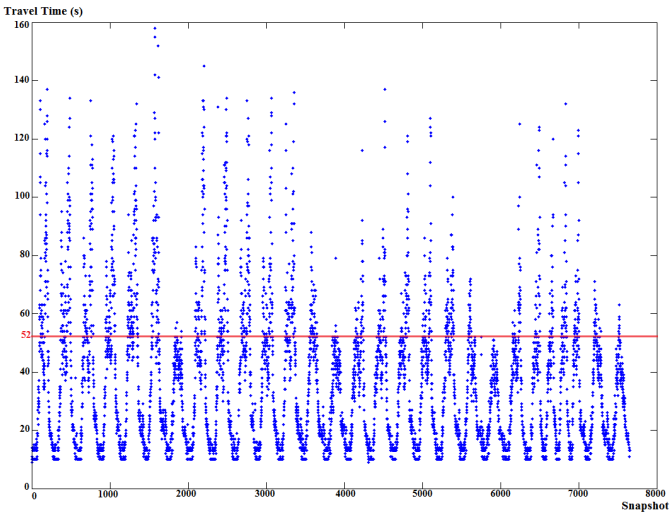


Figure 2: The average travel time within a segment in the network over 4 weeks. The red line represents the 80th percentile.

Figure 2 shows the distribution of average travel time within one segment over 4 weeks data. The 80th percentile boundary is presented by the red line (52 seconds) which is used as boundary for classification of segment's congestion status for each snapshots. This segment will be marked as congested in 20 percent of the snapshots which have recorded the average travel time longer than 52 seconds.

At the end of this step, the list of all congested segments in the network for each snapshot are identified and they will be used as inputs to construct the Congestion Trees.

4.3 Constructing Congestion Trees

This section presents an algorithm named STCTree that finds congestions dependencies by looking at the relationship of congestions from the earliest snapshot through the last.

Algorithm 1 STOTree: constructing all congestion trees

Input: STC: a set of spatial-temporal congestions of size t x k where t is the number of snapshot, and k is the number of congestions to examine in a snapshot.
Output: STOTrees: a list of roots of spatial-temporal trees.

1: $STOTrees = empty$
2: **for** Each snapshot $S_i (i \in (1, ..., t))$ **do**
3: **for** Each congestion $j (j \in (1, ..., k))$ in S_i **do**
4: $STORoot_{i,j}$ = FindAllChildren$(STC_{i,j}, i)$;
5: $STOTrees = STOTrees \cup STORoot_{i,j}$;
6: **end for**
7: **end for**
8: Return $STOTrees$;
Subroutine: FindAllChildren$(STC_{i,j}, i)$
9: **if** S_i is the last snapshot **then**
10: Return $STC_{i,j}$
11: **end if**
12: $STC_{i,j}.subnodes = empty$
13: **for** Each congestion $u (u \in (1, ..., k))$ in STC_{i+1} **do**
14: **if** $STOTrees$ contains $STC_{i+1,u}$ **then**
15: continue
16: **end if**
17: **if** $STC_{i,u}.des = STC_{i+1,j}.src$ **then**
18: $STC_{i,j}.subnodes = STC_{i,j}.subnodes \cup FindAllChildren(STC_{i+1,u}, i+1)$
19: **end if**
20: **end for**
21: Return $STC_{i,j}$;

The main insight of STCTree is that an congestion STC_1 is a parent of another congestion STC_2 if STC_1 occurred before STC_2 in time and they are spatially correlated.

Algorithm 1 is the previous implementation for constructing congestion causal trees [17]. All possible descendants of a node was retrieved by a recursive function which is called on each congestion of the current snapshot to compare with each congestion of next snapshot. The authors outlined that the overall time complexity of the congestion tree construction process on each snapshot is upper bounded by $O(n^2)$, where n is the number of congestions in a snapshot. Firstly, this is just the upper bound for the number of comparisons between destinations of current congestions and the sources of next congestions while the complexity of recursively constructing all subtrees for next snapshot was not considered. Secondly, a lot of repeated work can be observed. For example, the construction of trees in snapshot 1 requires the trees in all following snapshots to be computed. Then the same processes are repeated for snapshot 2, 3, ...t. As a result, the complexity of this algorithm can be exponential in the worst case as demonstrated below.

Denote by C_i $(i = 1, 2, ...t)$ the number of congestions and by $S(i)$ the number of comparisons required at snapshot i for constructing the tree. In the worst case, suppose $C_i \geq n > 0$ and most connections are made at every snapshot.

The last snapshot contains C_t congestions that need to be checked with previous snapshot:

$$S(t) = C_t \tag{1}$$

The recursive call (lines 13 to 19) is represented as:

$$S(i) = C_i \times S(i+1), i = 1, 2, ...t-1 \tag{2}$$

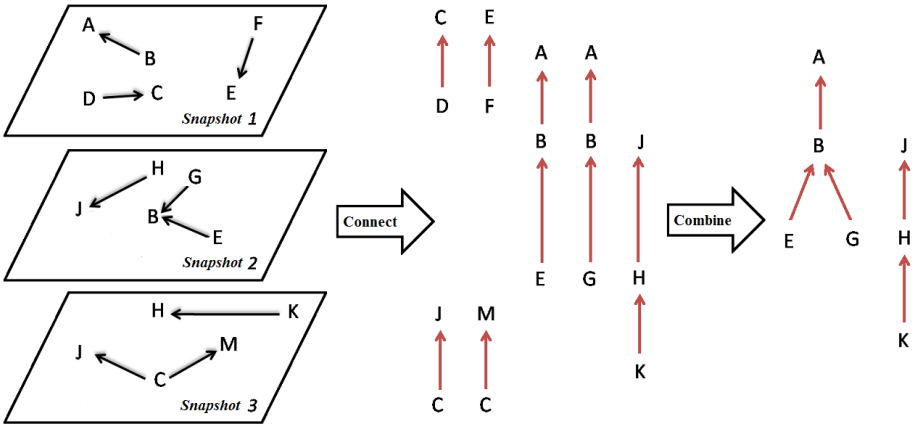


Figure 3: An example for demonstrating the process of building a forest of two congestion trees. The subfigure on the left illustrates top 3 congested segments in three consecutive snapshots, and the one on the right shows two congestion trees obtained from these snapshots.

$$S(i) = \prod_{k=i}^{t} C_k \tag{3}$$

Then the total comparisons for Algorithm 1 is:

$$S = \sum_{i=1}^{t-1} S(i) = \sum_{i=1}^{t-1} \prod_{k=i}^{t} C_k \tag{4}$$

Because there are at least n congestions at every snapshot:

$$S > \sum_{i=1}^{t-1} n^i = \frac{n^t - 1}{n - 1} - 1. \tag{5}$$

Consequently, the worst case lower bound complexity of the recursive algorithm is exponential time $\Omega(n^{t-1})$. In this paper, we proposed an dynamic programming implementation of the above algorithm which is always executed in polynomial time. Furthermore, the number of comparisons required to construct all the tree is minimal and not depend on the number of connections or size of congestion trees. In other words, the complexity of the present algorithm is only depended on the number of detected congestions from each snapshot.

Algorithm 2 demonstrates the process of constructing congestion trees from discovered congested segments. As seen from the algorithm (line 3), the trees are constructed bottom-up starting from last snapshot S_t to previous snapshots. Hence, at a specific snapshot S_i, all the potential subtrees in the following snapshot S_{i+1} are generated. In case we just need to compute the trees for a specific snapshot S_i, the space can be optimized by storing the previously considered snapshot S_{i+1} only because that is all we need to complete the snapshot S_i in the time frame. From now, the term previous snapshot will refer to the previously considered snapshot from the algorithm.

If there is no congested segments detected at a snapshot, the forest is set to empty (lines 5-9). When there is no tree in the previous snapshot, the forest contains a list of single root trees (roots are congestions in the current snapshot (lines 10-14)). These single node trees will then be removed if there is no later connections to them (line 32). Each tree is represented by a list of connected segments where first element is the root (latest segment that has been connected to the tree). Because the comparisons are only made between the roots and congestions, this simple data structure allows easy retrieval of the root from the tree (lines 17-23). Each subtree is built only once and will be removed later when there is at least one link to the congestions in the preceding snapshot (line 21 and 29). Then for each snapshot, the constructed forest is combined by grouping of all trees that share the same root into single trees (line 27).

$$S = \sum_{i=1}^{t-1} S(i) = \sum_{i=1}^{t-1} C_i \times C_{i+1} \tag{6}$$

Then the average complexity is:

$$S = \sum_{i=1}^{t-1} n \times n = t \times n^2 = O(tn^2) \tag{7}$$

Now we give an example by using Figure 3 to demonstrate the process of Algorithm 2 for building congestion trees. Figure 3 uses top 3 congestions in three consecutive snapshots, so the input parameters in Algorithm 2 in this case is t = 3. The algorithm starts from snapshot 3 (line 3) with forest is a list of single root trees, e.g. C → J, C → M, and K → H (lines 10-14). Then for each of the three congestions in snapshot 2 (line 17), i.e., H → J, G → B, and E → B, the algorithm searches in snapshot 3 and checks whether there is any root that can be a child of these congestions (lines 19 to 24). This allows the algorithm to find segment K → H as children of H → J. Then similarly it identifies links E → B and G → B in time frame 2 as a child of B → A in time frame 1. As can be seen from snapshot 1, there are two trees with the same root congestion, e.g E → B → A and G → B → A.

After combining trees with same root segment and removing single segment trees, two congestion trees are built up as shown in the right side of Figure 3. In this way, Algorithm 2 scans through all snapshots of traffic data, and builds a forest of various congestion trees. The result causal trees are interpreted as:

- At Snapshot 1, there was a congestion on the route from site A to site B. Until snapshot 2, this congestion

Algorithm 2 STCTree: constructing all congestion trees

Input: STC: a set of spatial-temporal congestions of size t where t is the number of snapshots, each snapshot contains a list of congestions to be examined.
Output: STCTrees: a list of forests of size t where t is the number of snapshots and each forest contains a list of spatial temporal trees.

```
1: STCTrees = empty
2: previous_snapshot = empty
3: for i = t downto 1 do
4:    congestions = STC[i]
5:    if congestions == empty then
6:       STCTrees[i] = empty
7:       previous_snapshot = empty
8:       continue
9:    end if
10:   if previous_snapshot == empty then
11:      STCTrees[i] == congestions
12:      previous_snapshot = congestions
13:      continue
14:   end if
15:   new_snapshot = empty
16:   used_trees = empty
17:   for each congestion in congestions do
18:      new_tree = [congestion]
19:      for Each tree in previous_snapshot do
20:         if congestion connectedto tree[1] then
21:            used_trees.append(tree)
22:            new_tree.append(tree)
23:         end if
24:      end for
25:      new_snapshot.append(new_tree)
26:   end for
27:   new_snapshot = combine(new_snapshot)
28:   STCTrees[i] = new_snapshot
29:   STCTrees[i + 1] = previous_snapshot − used_trees
30:   previous_snapshot = new_snapshot
31: end for
32: STCTrees = STCTrees − singlenodetrees
33: Return STCTrees
```

was clear however the congestions were transferred to two neighbour segments where the traffic flows to the previous congested segment, e.g from site E to site B and from site G to site B.

- Similarly, the travel time on segment H to J was recorded as very slow at Snapshot 2 then the slow pattern was moved to its preceding segment K to H.

4.4 Causal Congestion Detection

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

In the Frequent Subtree algorithm introduced by [17], they first find all single nodes whose supports exceed ϵ. Then the candidate subtrees are building up by checking whether each not in the set can be inserted in to the current trees (denoted by the root). To be completeness, this implementation requires that the parent nodes should be considered before the insertion of their children. However, the tree order is not reserved in the frequent single nodes list. An example of when this algorithm fails to generate the correct subtree is given in section 5.4.

Algorithm 3 AprioriSubtree: discovering frequent subtrees from STCongestion trees

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

```
1: frequentSubtrees = empty
2: frequentsets = Apriori(STCTrees, ϵ)
3: for Each frequentset in frequentsets do
4:    subtrees = construct_subtrees(frequentset)
5:    frequentSubtrees = frequentSubtrees ∪ subtrees
6: end for
7: Return frequentSubtrees;
   Subroutine: construct_subtrees(frequentset)
8: subtrees = [set(node) for node in frequentset]
9: while length(subtrees) > 1 do
10:   for tree T_i in subtrees do
11:      for tree T_j in subtrees[i+1:end] do
12:         if connected(T_i, T_j) then
13:            subtrees = subtrees - T_i - T_j + (T_i ∪ T_j)
14:            continue WHILE
15:         end if
16:      end for
17:   end for
18:   break
19: end while
20: subtrees = subtrees - singlenodetrees
21: return subtrees
```

As described in Section 4.3, each tree is presented by a list of connected segments. These lists can be considered as item sets where the Apriori algorithm is applied to identify the frequent sets [1], [16]. However, there is no constraint on original Apriori algorithm to guarantee the elements in the frequent set are all connected to form the subtree. In this case, each frequent set can comprise of several subtrees. Hence, we propose an post-processing step to Apriori algorithm to deal with the problem of finding frequent subtrees rather than frequent item sets.

The process of discovering frequent substructures from constructed congestion trees is shown in Algorithm 3. Given a predefined support threshold ϵ, the algorithm first finds all item sets whose supports exceed ϵ (line 2), then this set of segments is used to reconstruct the component subtrees (lines 4). For each pair of trees in the current list, the algorithm checks whether they are "connected" to each other (line 12). The two trees are "connected" if there exists a link between a segment of first tree and a segment of second tree or vice versa. When a combination is made, the length of tree list is reduced by 1 (line 13) and the process is restarted immediately (line 14). The construct_subtrees subroutine combines (union) the connected segments together until no further combination can be made or there is only one tree left (line 8-21). As a result, this routine outputs a list of subsets (subtrees) that satisfies the two conditions below:

- There exists a route between any two segments in the same subset.
- There is no link between any two segments in different subsets.

Because each subtree belongs to at least one item set whose support has already exceed ϵ, the subtree's support rate definitely exceeds the given support threshold (correctness). Furthermore, each subtree can be represented by one item set, the frequent items sets should cover all frequent subtrees that satisfy the support threshold ϵ. As a result, the algorithm guarantees to generate all possible frequent subtrees given ϵ (completeness).

5. MODELLING CONGESTION PROPAGATION

In this section, the approach of modelling the congestion propagation network using DBN is presented. We then proposed a practical method to calculate the distribution of a given congestion propagation structure.

5.1 A Dynamic Bayesian Network Approach

DBN is a popular approach for modelling spatial-temporal data [11]. A DBN is a BN which associates variables to each other over consecutive time frames (snapshot). This network is usually referred as a 2-Time slice BN (2TBN) because it states that at any given time T, the value of a variable can be computed from the internal regressors and the immediate prior value (time T-1). The idea of DBN is closely similar to the real-world phenomenon of traffic congestion propagation where the condition of a segment at a specific snapshot S can be determined by the previous conditions of its connecting segments. However, to the best of our knowledge, there is no research on modelling traffic congestion propagation using DBN.

To construct the DBN congestion network, the segments are represented in term of a set of N_h random variables, $Q_t^{(i)}$, $i \in \{1, ..., N_h\}$, each of which can take on 2 possible values, $Q_t^{(i)} \in \{0, 1\}$ where 1 means segment i is congested at time t. The observation can be represented in terms of N_0 random variables [20].

In a DBN, the transition (denoted as $B_{\rightarrow}$) and observation models are then defined as a product of the conditional probability distribution (CPD) in the 2TBN:

$$P(Z_t|Z_{t-1}) = \prod_{i=1}^{N} P(Z_t^{(i)}|Pa(Z_t^{(i)})) \tag{8}$$

where $Z_t^{(i)}$ is the ith node in snapshot t and $Pa(Z_t^{(i)})$ are the parents of $Z_t^{(i)}$, which may be in the current or previous snapshot. The unconditional initial state distribution, $P(Z_1^{1:N})$ can be presented using a standard BN, namely B_1. Together, B_1 and $B_{\rightarrow}$ define the DBN.

Figure 5 illustrates the process of constructing the DBN for analysing the congestion propagation for a traffic network with three connected segments. Suppose we have a simple traffic network which comprises of three segments: EB and GB are connected to BA. As EB and GB both lead to BA, when BA is congested, it becomes the potential cause for congestions at EB and GB in the next time frame. Consequently, the causal links from BA to EB and GB are transferred over the two consecutive time frames. Furthermore, the status of each segment usually depends on its previous condition so the link between the same segment is added over the time. In this example, $Pa(BE_{t-1})$ includes three segments, e.g. BA_{t-2}, BE_{t-2} and BA_{t-1}.

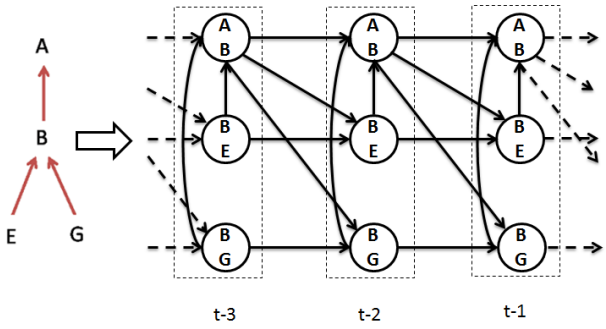


Figure 5: Modelling congestion propagation of a traffic network by DBN.

After modelling the congestion traffic network, all the general inference and learning can be performed based on DBN. The following section describes the use of DBN to calculate the distribution of congestion propagation.

5.2 Parameter Learning for Completed Data Set

The above modelling method describes the construction of the congestion propagation DBN given the static traffic network. This section focuses on the problem of estimating maximum likelihood (ML) parameters for a model given the structure and completed data.

Assume a data set of independent and identically distributed observed segments $D = \{Z^{(1)}, ..., Z^{(N)}\}$, then the likelihood of the data set is:

$$P(D|\theta, M) = \prod_{i=1}^{N} P(Z^{(i)}|\theta, M) \tag{9}$$

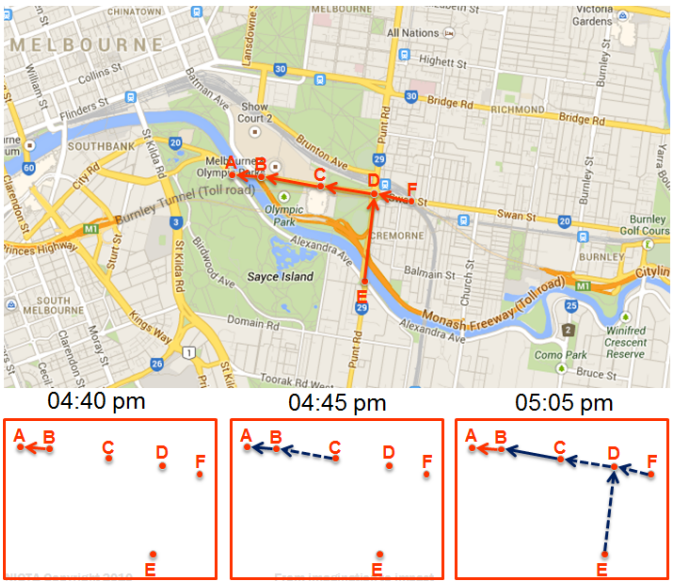
where M is implicit conditioning on the known structure of the model. For the representation convenience, M will be dropped from the following equations. The ML parameters are obtained by maximizing the log likelihood:

$$\ell(\theta) = \sum_{i=1}^{N} \log P(Z^{(i)}|\theta). \tag{10}$$

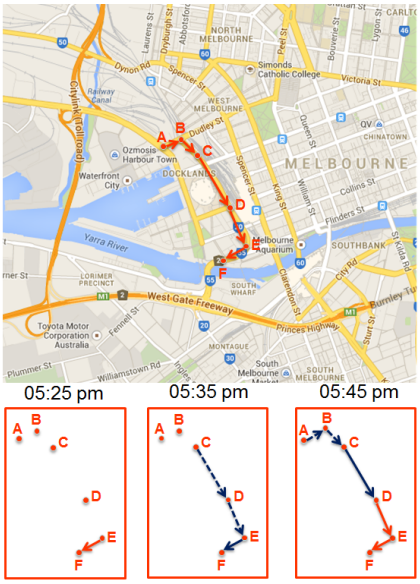
In case the observation includes all the segments in the network, then each term in the log likelihood further factors as:

$$\begin{aligned} \log P(Z^{(i)}|\theta) &= \log \prod_{j} P(Z_t^{(i)}|Pa(Z_t^{(i)}), \theta_t) \\ &= \sum_{t=1}^{T} \log P(Z_t^{(i)}|Pa(Z_t^{(i)}), \theta_t) \end{aligned} \tag{11}$$

where θ_t is the parameter that define the conditional probability of Z_t given its parents. Hence, the likelihood is decomposed into local terms involving *each segment and its parents*, simplifying the ML estimation problem [11]. For instance, θ_t is the conditional probability table for Z_t given



(a) The frequent subtree covering the "Olympic Park".



(b) The frequent subtree covering the "Etihad Stadium".

Figure 4: Top two frequent subtrees (frequency of 10 over 4 weeks) and areas they cover ("Olympic Park" and "Etihad Stadium"). Each subtree is visualised on the map with additional information of where and when the congestion is transferred to its surrounding area.

its parents, then the ML estimate of θ_t is simply a normalised table containing counts of each setting of Z_t given each setting of its parents in the data set.

5.3 Distribution Estimation for Congestion Trees

This section proposes the method for calculating the probability for the causal trees. After constructing the causal frequent trees, the joint distribution for a known-structure tree which includes T consecutive snapshots (slices) can be obtained by "unrolling" the network until we have T slices, and then multiplying together all of the CPDs:

$$P(Z_{1:T}^{(1:N)}) = \prod_{i=1}^{N} P_{B_1}(Z_1^{(i)}|Pa(Z_t^{(i)}) \times \prod_{t=2}^{T}\prod_{i=1}^{N} P_{B_\rightarrow}(Z_t^{(i)}|Pa(Z_t^{(i)})) \quad (12)$$

6. EXPERIMENTS AND ANALYSIS

In this section we report on the experiments carried out on the road network of a major Australian city. Our experiments are conducted on a 64 bit server with 3.4 GHz CPU and 16 GB memory. Although road traffic data was utilised in this experiments, the present methods and algorithms can be easily adapted into other domains such as finding bottle necks in the internet traffic data and water pipe data.

6.1 Data

The algorithms are tested based on a real travel time records generated by the road sensors in a period of 4 weeks (from 17/06/2013 to 14/07/2013). There are 281 sites and 586 segments in the examined network. The average sampling interval is 5 minutes and the total number of recorded snapshots over 4 weeks is 8064.

The current experimental data just covers the main roads of the Victoria traffic network where the sensors are located. However, with the quadratic running time for trees construction on each snapshot, our proposed method is applicable to more detailed and complicated networks.

6.2 Experiment on Congestion Trees

In this experiment, the value of percentile used to identify the congested segments was bounded between 50% and 90%, then its effects on constructing congestion trees were evaluated.

The minimum size of a tree (i.e. total number of segments) was set to 2, and hence singles segments (trees of size 1) were ignored in counting final congestion trees. The results from Figure 6 shows that although the maximum numbers of trees increase substantially when the percentile decreases, the maximum size of all trees has a smoothly increasing trend from 8 to 15 segments. With lowest percentile threshold of 50% for detecting congestions, the maximum number of trees in one snapshot is 66 and the largest tree contains 15 segments. Hence, the maximum number of consecutive snapshots involved is also 15 and the maximum duration for the congestion propagation is 75 minutes (15 snapshots x 5 mins). This observation validates our earlier belief that congestions caused by one single accident normally do not last very long, and that the maximum size of trees is usually small.

In Section 4.3, the time complexity of the congestion tree construction algorithm STCTree for each snapshot was proved to be upper bounded by n^2 where n is the maximum number of congestions in a single snapshot. Because n increases almost linearly when percentile is reduced, the average time used for building trees increases almost quadratically with n. Consequently, STCTree can potentially be used in an online setting to detect congestions causalities on the fly.

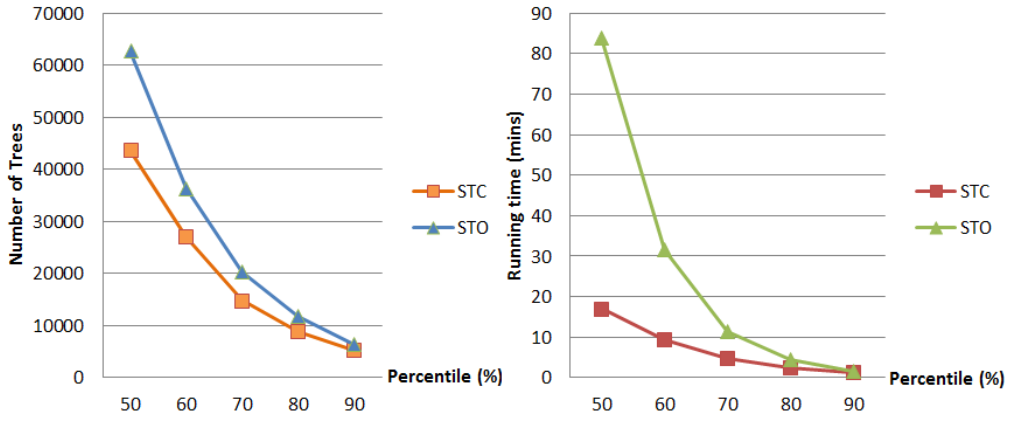


Figure 6: Comparison of STOTree and STCTree algorithms in the number of constructed trees and running time.

6.3 Experiment on Frequent SubTrees

The route indicated by the most frequent (i.e. with highest support) subtrees are the ones that have strategic design drawbacks from the perspective of urban road network planning. For example, the top two frequent subtrees (using 80th percentile) with at least 5 connected segments are both located in the CBD as shown in Figure 4. They are both occurred 10 times during weekday's rush hours in the period of 4 weeks.

Given the congestion was first happened at the root segment BA, we then modelled the structure of the congested tree using DBN as describe in Section 3 and estimated the joint distribution using Equation 10. The propagation probability of the first frequent tree is 2.24% and the second one is 0.77%. These probabilities are expected to be higher during the rush hours. The same experiment was run with the samples limited to afternoon rush hours between 4pm and 6pm. The results increased to 4.26% and 3.42% respectively. These probabilities seem small as compared to a pair of propagation but are relatively high with a 5-segment tree formation. It is really difficult for the propagation pattern to follow the exact order within fixed period of time. These subtrees indicate that both of the two areas were more frequently overloaded with vehicles and there may have potential design flaws in the current road network spanning.

7. DISCUSSIONS

In this paper, the proposed method for mining the congestion propagation patterns is a combination of STC and DBN. Each method individually has its own advantages and limitations which are discussed in this section.

The proposed dynamic programming implementation of STC algorithm has a running time of $O(n^2)$ where n is the maximum number of segments within a single snapshot. This makes STC efficient enough for discovering frequent sub-structures within large spatial-temporal traffic networks.

The frequent congestion subtrees algorithm is mainly based on the support rate which is calculated by divide the subtree's frequency to the total number of causal trees. The subtree is selected if its support rate exceeds the given support threshold ϵ. Hence, it requires the construction of all possible causal tree within the whole network to estimate the subtree's support rate. The support rate cannot reflect the probability for the forming of a tree given the congestion at the root segment.

Modelling the congestion propagation using DBN is a general method to describe the "nature" of the congestion propagation. The model can be used for inference and joint distribution estimation with both completed and hidden data. In our method, DBN model is used to support the STC by providing the congestion propagation distribution of a frequent subtree. For any given sub-structure with congestion at the root segment, the DBN's joint distribution estimation is solely depended on the observations of the involved segments rather than requiring data from the whole large network.

Discovering the most frequent or most probable sub-structures in DBN may require learning the whole network. Because of several NP-hardness results on learning BN, many algorithms for learning DBN are approximate, that employ either local search such as greedy hill-climbing, or a meta optimization framework such as genetic algorithm or simulated annealing [28, 21]. These algorithms are not efficiently in term of running time given a large network.

Our proposed method can utilise the advantages and overcome the disadvantages of individual approach by:

- Use the STC algorithm to discover the frequent causal trees which is applicable for the large network.
- Use DBN to calculate the distribution of congestion propagation from the small sub-structures which were discovered by STC.

8. CONCLUSION

In this paper, the problem of detecting spatio-temporal congestions and their causal interactions from traffic data streams has been studied. We have proposed STCTree, the algorithm for discovering spatio-temporal congestions and causal relationships between them. Furthermore, the frequent subtree algorithm can be used to reveal recurrent in the road network. Based on the STCTree and frequent subtree algorithm we were able to identify real and valid instances of congestions propagations in network traffic data. This suggests that our approach has the potential of contributing to a new data driven approach towards road traffic analysis.

9. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Data Bases*, pages 478–499. Morgan Kaufmann, Los Altos, CA, 1994.

[2] Y. Bu, L. Chen, A. W.-C. Fu, and D. Liu. Efficient anomaly monitoring over moving object trajectory streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 159–168. ACM, 2009.

[3] H. Cao, N. Mamoulis, and D. W. Cheung. Mining frequent spatio-temporal sequential patterns. In *Fifth IEEE International Conference on Data Mining (ICDM)*, pages 8–pp. IEEE, 2005.

[4] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

[5] V. W. Chu, R. K. Wong, W. Liu, and F. Chen. Causal structure discovery for spatio-temporal data. In *Database Systems for Advanced Applications*, pages 236–250. Springer, 2014.

[6] C. F. Daganzo. The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research Part B: Methodological*, 28(4):269–287, 1994.

[7] C. F. Daganzo. The cell transmission model, part ii: network traffic. *Transportation Research Part B: Methodological*, 29(2):79–93, 1995.

[8] C. F. Daganzo and J. A. Laval. Moving bottlenecks: A numerical method that converges in flows. *Transportation Research Part B: Methodological*, 39(9):855–863, 2005.

[9] K. Fukuda, H. Takayasu, and M. Takayasu. Spatial and temporal behavior of congestion in internet traffic. *Fractals*, 7(01):23–31, 1999.

[10] D. C. Gazis and R. Herman. The moving and "phantom" bottlenecks. *Transportation Science*, 26(3):223–229, 1992.

[11] Z. Ghahramani. Learning dynamic bayesian networks. In *Adaptive processing of sequences and data structures*, pages 168–197. Springer, 1998.

[12] B.-H. Juang. Hidden markov models. *Encyclopedia of Telecommunications*, 1985.

[13] J.-G. Lee, J. Han, and X. Li. Trajectory outlier detection: A partition-and-detect framework. In *24th IEEE International Conference on Data Engineering (ICDE)*, pages 140–149. IEEE, 2008.

[14] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604. ACM, 2007.

[15] M. Lippi, M. Bertini, and P. Frasconi. Collective traffic forecasting. In *Machine Learning and Knowledge Discovery in Databases*, pages 259–273. Springer, 2010.

[16] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Fourth International Conference on Knowledge Discovery and Data Mining*, pages 80–86. AAAI Press, 1998.

[17] W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xing. Discovering spatio-temporal causal interactions in traffic data streams. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1010–1018. ACM, 2011.

[18] J. Long, Z. Gao, H. Ren, and A. Lian. Urban traffic congestion propagation and bottleneck identification. *Science in China Series F: Information Sciences*, 51(7):948–964, 2008.

[19] A. C. Lozano, H. Li, A. Niculescu-Mizil, Y. Liu, C. Perlich, J. Hosking, and N. Abe. Spatial-temporal causal modeling for climate change attribution. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 587–596. ACM, 2009.

[20] K. P. Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, 2002.

[21] A. Nägele, M. Dejori, and M. Stetter. Bayesian substructure learning-approximate learning of very large network structures. In *Machine Learning: ECML 2007*, pages 238–249. Springer, 2007.

[22] G. F. Newell. A simplified theory of kinematic waves in highway traffic, part i: General theory. *Transportation Research Part B: Methodological*, 27(4):281–287, 1993.

[23] G. F. Newell. A moving bottleneck. *Transportation Research Part B: Methodological*, 32(8):531–537, 1998.

[24] D. Ni and J. D. Leonard II. A simplified kinematic wave model at a merge bottleneck. *Applied mathematical modelling*, 29(11):1054–1072, 2005.

[25] L. X. Pang, S. Chawla, W. Liu, and Y. Zheng. On detection of emerging anomalous traffic patterns using gps data. *Data & Knowledge Engineering*, 87:357–373, 2013.

[26] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[27] N. Pelekis, I. Kopanakis, C. Panagiotakis, and Y. Theodoridis. Unsupervised trajectory sampling. In *Machine Learning and Knowledge Discovery in Databases*, pages 17–33. Springer, 2010.

[28] N. X. Vinh, M. Chetty, R. Coppel, and P. P. Wangikar. Globalmit: learning globally optimal dynamic bayesian network with the mutual information test criterion. *Bioinformatics*, 27(19):2765–2766, 2011.