Traffic Density-Based Discovery of Hot Routes in Road Networks*

Xiaolei Li, Jiawei Han, Jae-Gil Lee, and Hector Gonzalez

University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Abstract. Finding hot routes (traffic flow patterns) in a road network is an important problem. They are beneficial to city planners, police departments, real estate developers, and many others. Knowing the hot routes allows the city to better direct traffic or analyze congestion causes. In the past, this problem has largely been addressed with domain knowledge of city. But in recent years, detailed information about vehicles in the road network have become available. With the development and adoption of RFID and other location sensors, an enormous amount of moving object trajectories are being collected and can be used towards finding hot routes.

This is a challenging problem due to the complex nature of the data. If objects traveled in organized clusters, it would be straightforward to use a clustering algorithm to find the hot routes. But, in the real world, objects move in unpredictable ways. Variations in speed, time, route, and other factors cause them to travel in rather fleeting "clusters." These properties make the problem difficult for a naive approach. To this end, we propose a new density-based algorithm named FlowScan. Instead of clustering the moving objects, road segments are clustered based on the density of common traffic they share. We implemented FlowScan and tested it under various conditions. Our experiments show that the system is both efficient and effective at discovering hot routes.

1 Introduction

In recent years, analysis of moving object data [7] has emerged as a hot topic both academically and practically. In particular, the tracking of moving objects in road networks is becoming quite popular. GPS devices embedded in vehicles or RFID sensors on the streets can track a vehicle as it moves throughout the city traffic grid. There are many useful applications with such data. For instance, the OnStar system in General Motors vehicles notifies police of the vehicle's GPS location when a crash is detected. E-ZPass sensors (using RFID technology) automatically pay tolls so traffic is not disturbed. GPS navigation systems offer driving directions in real-time. On a more aggregate level, average speeds or

* The work was supported in part by Boeing company and the U.S. National Science Foundation NSF IIS-05-13678/06-42771, and NSF BDI-05-15813. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

D. Papadias, D. Zhang, and G. Kollios (Eds.): SSTD 2007, LNCS 4605, pp. 441–459, 2007.

traffic density is used to update driving time estimates in real-time or warn police of potential problem areas.

In this work, we address the problem of discovering *hot routes* in a road network. Informally, a hot route is a general traffic flow pattern. For example, "Many people in Oakland travel westbound on the Bay Bridge to reach downtown San Francisco at 7:30am." The set of hot routes offers direct insight into the city's traffic patterns. City officials can use them to improve traffic flow. Store owners and advertisers can use them to better position their properties. Police officials can use them to maximize patrol coverage.

Example 1. Figure 1(a) shows live traffic data[1] in the San Francisco Bay Area on a weekday at approximately 7:30am local time. Different colors show different levels of congestion (*e.g.*, red/dark is heavy congestion). `511.org` in the Bay Area gathers such data in real-time from RFID transponders located inside vehicles[2]. A likely hot route in Figure 1(a) is $A \rightarrow B$ (*i.e.*, highway CA-101). A is near the San Francisco International Airport. B is near the San Mateo Bridge. Figure 1(b) shows a closeup view of location B. Three additional locations x, y, and z are shown. Without actually observing the flow of traffic, it is unclear whether $y \rightarrow x$ is a hot route, or $y \rightarrow z$, or $x \rightarrow z$. FlowScan aims to solve this problem. ■

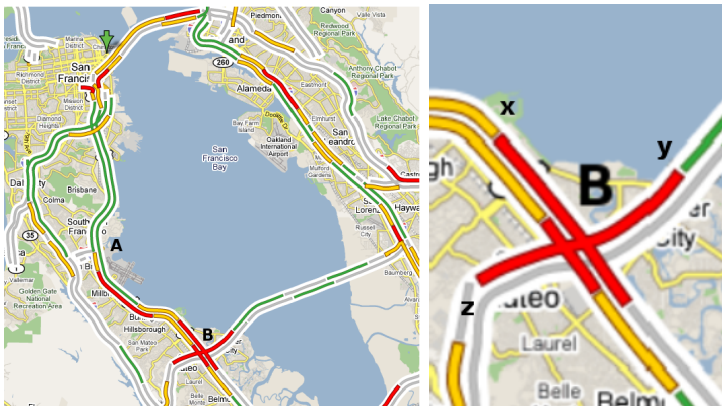


Fig. 1. Snapshot of San Francisco Bay Area traffic

At first glance, this may seem like an easy problem. A quick look at Figure 1(a) shows the high traffic roads in red. With some domain knowledge, we know that San Francisco, Oakland, and other densely populated regions are likely to be sources and destinations of traffic since many people live and work there. However, such domain knowledge is not always available. Additionally,

[1] http://maps.google.com
[2] http://www.bayareafastrak.org

real world traffic is a very *complex* data source. Objects do not travel in organized clusters. Two objects traveling from the same place to another place may take just slightly different routes at slightly different speeds and times. Random traffic conditions, such as a traffic accident or a traffic light, can cause even more deviations. Furthermore, hot routes do not have to be disjoint. Highways or major roads are popular pathways and several hot routes can share them. As a result, the mining algorithm must be robust to the variations within a hot route and amongst a set of hot routes.

We now state our problem as follows: *Given a set of moving object trajectories in a road network, find the set of hot routes.* A road network is represented by a graph $G(V, E)$. E is the set of directed edges, where each one represents the smallest unit of road segment. V is the set of vertices, where each one represents either a street intersection or important landmark. T is the set of trajectories, and each trajectory consists of an ID (tid) and a sequence of edges that it traveled through: $(tid, \langle e_1, \ldots, e_k \rangle)$, where $e_i \in E$. Objects can only move on E and must travel the entirety of an edge. T is assumed to be collected from a similar time window; otherwise, different time windows might blur meaningful hot routes.

Informally, a *hot route* is a general path in the road network which contains heavy traffic. It represents a general flow of the objects in the network. Formally, it is a sequence of edges in G. The edges need not to be adjacent in G, but they should be near each other. Further, a sequence of edges in a hot route should *share a high amount of traffic between them.*

The rest of the paper is organized as follows. Section 2 gives an overview of the proposed solution and also alternative approaches. Section 3 lists some typical traffic behaviors and how they can confuse the naïve approaches. Section 4 describes the algorithm. Section 5 shows experiment results. Related work is discussed in Section 6. And finally, we conclude the study in Section 7.

2 Solution Overview

FlowScan extracts hot routes using the density of traffic on edges and sequences of edges. Intuitively, an edge with heavy traffic is potentially a part of a hot route. Edges with little or no traffic can be ignored. Also, two near-by edges that share a high amount of traffic between them are likely to be a part of the same hot route. This implies that the objects traveled from one edge to the other in a sequence. And lastly, a chain of such edges is likely to be a hot route.

We also list some possible alternative methods from related fields.

Alternative Method 1: Moving object clustering [6] discovers groups of objects that move together. The trajectory of each cluster can be marked as a hot route. We call this class of approaches AltMoving.

Alternative Method 2: Simple graph linkage is another possible approach. One could gather all the edges in G with heavy traffic and connect them via their graph connectivity. Then, each connected component is marked as a hot route. We call this class of approaches AltGraph.

Alternative Method 3: Trajectory clustering [10] discovers groups of similar sub-trajectories from the whole trajectories of moving objects. Each resultant cluster is marked as a hot route. We call this class of approaches AltTrajectory.

FlowScan and the three alternative methods offer very different approaches to the same data. One could view them in a spectrum. At one end of the spectrum are AltMoving and AltTrajectory where attention is paid to the individual objects. This is helpful in problems where the goal is to identify behaviors of individuals. At the other end of the spectrum is AltGraph where attention is paid to the aggregate. That is, objects' trajectories are aggregated into summaries and analysis is performed on the summary. This is helpful in problems where the goal is to learn very general information about the data. FlowScan can be viewed as an intermediate between these two extremes. The behaviors of the individuals (specifically, the common traffic between sequences of edges) are retained and affect high-level analysis about aggregate behavior.

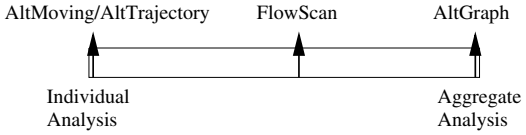


Fig. 2. Spectrum view of FlowScan and alternative methods

3 Traffic Behavior in Road Networks

In this section, we list some common real world traffic behaviors and examine how FlowScan and the alternative approaches can handle them.

3.1 Traffic Complexity

A major characteristic of real world traffic is the amount of complexity. Instead of neat clusters, objects travel with different speeds and times even when they are on the same route. For example, in a residential neighborhood, many people leave for work in the morning and travel to the business district using approximately the same route. However, it is very unlikely that a group will leave at the same time and also travel together all the way to their destination. Various events (*e.g.*, traffic light) can easily split them up.

Algorithms in the AltMoving class will not work very well with such complex data. Clusters in the technical sense only last for a short period of time or short distance. The same is true for AltTrajectory if speed/time is encoded into the trajectories. These algorithms lack *aggregate analysis* and as a result, they are likely to find too many short clusters and miss the overall flow. FlowScan connects road segments by the amount of traffic they share. So even if the objects change slightly or if the objects do not travel in compact groups, the amount of common traffic between consecutive edges in a hot route will still be high.

3.2 Splitting/Joining Hot Routes

Figure 3 shows a sample city traffic grid. The shade on each road segment indicates the amount of traffic on it; the darker the shade, the heavier the traffic. Suppose the two correct hot routes are $A \rightarrow B \rightarrow C$ and $A \rightarrow B \rightarrow D$. Figure 3 shows a splitting of traffic at node B: some objects which moved from A to B go to C while others go to D. There are also other objects which move from C to D and vice-versa. This situation is very common in real world traffic. B could be the location in Chicago where I-90/94 splits into I-90 and I-94.

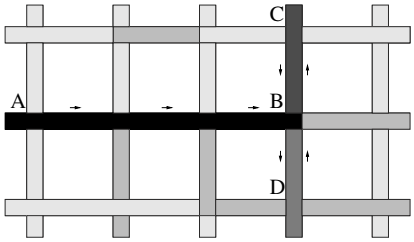


Fig. 3. Splitting hot routes: $A \rightarrow B \rightarrow C$ and $A \rightarrow B \rightarrow D$

With AltGraph, since all edges from A to C and D are connected, they would be incorrectly identified as a single big hot route. Notice that there is no *individual analysis* in AltGraph. With AltTrajectory, $A \rightarrow B$ and $B \rightarrow C$ will not be joined together because the physical similarity between them is low (*i.e.*, hard left turn). Likewise for $A \rightarrow B \rightarrow D$. This flaw exists for the joining of traffic as well. In other words, if the arrows in Figure 3 were reversed, it would illustrate the problem of two hot routes joining at B.

In FlowScan, edges $B \rightarrow C$ and $B \rightarrow D$ will not be connected directly because they do not share any traffic. This is because physically, objects have to choose between the two edges and cannot travel on both. Further, $A \rightarrow B$ will be connected to both $B \rightarrow C$ and $B \rightarrow D$ because it shares traffic with both.

3.3 Overlapping Hot Routes

In addition to splits or joins, two hot routes may overlap each other. Figure 4 shows an example with two distinct hot routes: $A \rightarrow B$ and $B \rightarrow C$. Situations like this are common. Suppose B is a parking garage used by nearby residents during the night and incoming workers during the day. In the morning, residents drive out of the parking garage ($B \rightarrow C$) and other people arrive from various locations to park ($A \rightarrow B$).

Consider how AltGraph will handle this situation. Since $A \rightarrow B$ and $B \rightarrow C$ are connected in G at node B, the two hot routes will be joined together incorrectly. This is due to the lack of *individual analysis* on the edges during linking. The same happens with AltTrajectory, though for a different reason. $A \rightarrow B$ and

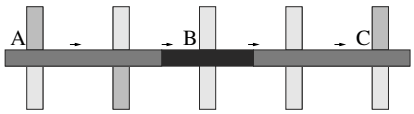


Fig. 4. Overlapping hot routes: $A \rightarrow B$ and $B \rightarrow C$

$B \rightarrow C$'s shapes are similar and will be clustered together. With FlowScan, consecutive edges within a hot route must share a minimum number of common objects. If such edges were parts of different hot routes, this condition will not be satisfied, and thus, a single erroneous hot route will not be formed.

3.4 Slack Within Hot Routes

Figure 5 shows a hot route with some slight slack. A hot route exists from A to B in the grid. At the intermediate locations, objects are faced with different choices in order to reach B. Suppose the choices are essentially equivalent in terms of distance and speed and that traffic is split equally between them.

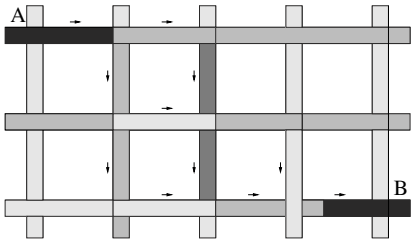


Fig. 5. Slack within a hot route: $A \rightarrow B$

Consider how AltMoving will handle this deviation. Suppose the distance between the equivalent paths is larger than the maximum intra-cluster distance. This will cause the cluster at A to break into several smaller clusters when it reaches B. Next, consider AltGraph. Suppose the partitioning of traffic reduced the density on the intermediate edges to be below the "heavy" threshold. This would break the graph connectivity condition and miss the hot route from A to B. A similar error could occur with AltTrajectory if the traffic becomes too diluted between A and B or if the shapes become too dissimilar. With FlowScan, edge connectivity in G is not a required condition. Edges are connected in the hot route if they share common traffic and if they are near each other. Here, as long as A is within a given distance from B, the hot route will remain intact.

4 Density-Based Hot Route Extraction

In this section, we will give formal definitions of FlowScan, which uses traffic density information in road networks to discover hot routes.

4.1 Traffic-Density Reachability

Definition 1 (Edge Start/End). *Given a directed edge r, let the $start(r)$ be the starting vertex of the edge and $end(r)$ be the ending vertex.*

Definition 2 (ForwardNumHops). *Given edges r and s, the number of forward hops between r and s is the minimum number of edges between $end(r)$ and $end(s)$ in G. It is denoted as $ForwardNumHops(r, s)$.*

Recall that G is directed. This implies that an edge that is incident to $start(r)$ in G will not have a $ForwardNumHops$ value of 0 unless it is also incident to $end(r)$.

Definition 3 (*Eps*-neighborhood). *The* Eps-neighborhood *of an edge r, denoted by $N_{Eps}(r)$, is defined by $N_{Eps}(r) = \{s \in E \mid ForwardNumHops(r, s) \leq Eps\}$ where $Eps \geq 0$.*

The *Eps*-neighborhood of r contains all edges that are within *Eps* hops away from r, *in the direction* of r. Semantically, this captures the flow of traffic and represents where objects are within *Eps* hops after they *exit* r. Figure 6 shows the 1-neighborhood of edge r.

Note that having the forward direction in the *Eps*-neighborhood makes the relation non-symmetric. In Figure 6 for example, the two edges in the circle are in the 1-neighborhood of r, but r is not in the 1-neighborhood of either of them. In fact, the only time when the *Eps*-neighborhood relation is symmetric is when two edges form a cycle within themselves. This is usually rare in road networks with one exception, and that is when one considers two sides of the *same* street. Figure 7 shows an example. In it, r_0 is in the 1-neighborhood of r_1 and vice-versa, because they form a cycle. This will happen for all two-way streets in the network. Though typically, an object will not travel on both sides of the same street within a trajectory. It could only happen with U-turns or if one end of the street is a dead end.

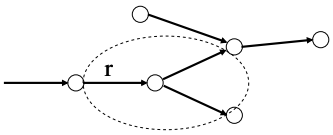


Fig. 6. 1-neighborhood of r

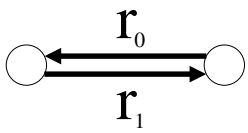


Fig. 7. Two sides of the same street

We did not choose a spatial distance function (*e.g.*, Euclidean distance), because the number of hops better captures the notion of "nearness" in a transportation network. Consider a single road segment in the transportation network. If it were a highway, it might be a few kilometers long. But if it were a city block in downtown, it might be just a hundred meters. However, since an object has to travel the entirety of that edge, the two adjacent edges are the same "distance" apart no matter how long physically that intermediate edge is.

Definition 4 (Traffic). *Let traffic*(r) *return the set of trajectories that contains edge* r*. Recall trajectories are identified by unique IDs.*

Definition 5 (Directly traffic density-reachable). *An edge* s *is* directly traffic density-reachable *from an edge* r *wrt two parameters, (1) Eps and (2) MinTraffic, if all of the following hold true.*

1. $s \in N_{Eps}(r)$
2. $|traffic(r) \cap traffic(s)| \geq MinTraffic$

Intuitively, the above criteria state that in order for an edge s to be directly traffic density-reachable from r, s must be near r, and $traffic(s)$ and $traffic(r)$ must share some non-trivial common traffic. The "nearness" between the two edges is controlled by the size of the *Eps*-neighborhood. This directly addresses the slack issues in Section 3.4. As long as the slack is not larger than *Eps*, two edges will stay directly connected via this definition.

The second condition of two edges sharing traffic is intuitive. It is also at the core of FlowScan. The flaw of methods in the AltGraph class is that aggregation on the edges has erased the identities of the objects. As a result, two edges with high traffic on them and near each other will look the same regardless if they actually *share* common traffic. By having the second condition rely on the common traffic, one can get a better idea of how objects actually move in the road network.

Directly traffic density-reachable is not symmetric for pairs of edges because the *Eps*-neighborhood is not symmetric. Though, for the same reasons that two edges might be in the *Eps*-neighborhood of each other, two edges could be directly traffic density-reachable from each other.

Definition 6 (Route traffic density-reachable). *An edge* s *is* route traffic density-reachable *from an edge* r *wrt parameters Eps and MinTraffic if:*

1. *There is a chain of edges* r_1, r_2, ..., r_n, $r_1 = r$, $r_n = s$, *and* r_i *is directly traffic density-reachable from* r_{i-1}.
2. *For every Eps consecutive edges (i.e.,* r_i, r_{i+1}, ..., r_{i+Eps}*) in the chain,* $|traffic(r_i) \cap traffic(r_{i+1}) \cap \ldots \cap traffic(r_{i+Eps})| \geq MinTraffic$.

Definition 6 is an extension of Definition 5. It states that two edges are route reachable if there is a chain of directly reachable edges in between and that if one were to slide a window across this chain, edges inside every window share common traffic. The sliding window directly address the overlapping behavior as described in Section 3.3. At the boundaries of two overlapping hot routes, the second condition will break down and thus break the overlapping hot routes into two. The *Eps* parameter is being reused here to control the width of a sliding window through the chain. The reuse is justified because their semantics are similar, but one could just as well use a separate parameter.

The reason for using a sliding window is based on our observation that a trajectory can contribute to only a portion of a hot route. This better matches real world hot route behavior. For example, a hot route exists from the suburb

to downtown in the morning. Figure 8 shows an illustration. However, most people do not travel the entirety of the hot route. More often, they live and work somewhere in between the suburb and downtown. But in the aggregate, a hot route exists between the two locations.

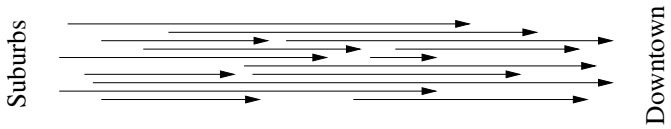


Fig. 8. Route traffic density-reachable

4.2 Discovering Hot Routes

The hot route discovery process follows naturally from Definition 6. It is an iterative process. Roughly, one starts with a random edge, expands it to a hot route, and repeats until no more edges are left. The question is then with which edge(s) should each iteration begin. To this end, we introduce the concept of a **hot route start**, which is the first edge in a hot route. Intuitively, an edge is a hot route start if none of its preceding directly traffic density-reachable neighbors are part of hot routes.

Definition 7 (Hot Route Start). *An edge r is a* hot route start *wrt MinTraffic if the following condition is satisfied.*

$$\left| \mathit{traffic}(r) \setminus \bigcup \{\mathit{traffic}(x)\} \right| \geq \mathit{MinTraffic}$$

where $\{x \mid \mathit{end}(x) = \mathit{start}(r) \wedge |\mathit{traffic}(x)| \geq \mathit{MinTraffic}\}$.

The question is whether all hot routes begin from a hot route start. The following lemma addresses this.

Lemma 1 (Hot Route Start). *Hot routes must begin from a hot route start.*

Proof: There are two ways for a hot route to begin on an edge r. The first way is when *MinTraffic* or more objects start their trajectory at r. In this case, none of these objects will appear in $\mathit{start}(r)$'s adjacent edges because they simply did not exist then. As a result, the set difference will return at least *MinTraffic* objects and thus marking r as a hot route start. The second way is when *MinTraffic* or more objects converge at r from other edges. The source of traffic on r is exactly the set of edges adjacent to $\mathit{start}(r)$. Suppose one of these edges, x, contains more than *MinTraffic* objects on it. In this case, x is part of another hot route, and the objects that moved from x to r should not contribute to r. However, if it does not contain more than *MinTraffic* objects, it cannot be in a hot route and its objects are counted towards r. If more than *MinTraffic* objects are counted towards r, then it is the start of a hot route. ∎

4.3 Algorithm

Definitions 6 and 7 form the foundation of the hot route discovery process. A simple approach could be to initialize a hot route to a hot route start and iteratively add all route traffic density-reachable edges to it. By repeating this process for all hot route starts in the data, one can extract all the hot routes. The question is then how to efficiently find all route traffic density-reachable edges given an existing hot route. If new edges are added in no particular order, then one would have to search through all existing edges in the hot route at every iteration. This is very inefficient. Further, if the hot route splits, it could become tricky if it is in the middle.

To alleviate this problem, we restrict the growth of a hot route to be only at the *last* edge. A hot route is a sequence of edges so the last edge is always defined. By growing the hot route at the end one edge at a time, only the *Eps*-neighborhood of the last edge needs to be extracted. This is much more efficient than extracting the *Eps*-neighborhoods of all edges in the hot route. This is still a complete search because all possible reachable edges are examined but just with some order. It is essentially a breadth-first search of the road network. Then for each neighboring edge, the *route* traffic density-reachability condition is checked against the last few edges of the hot route (*i.e.*, window). If the condition is satisfied, the edge is appended to the hot route; otherwise, the edge is ignored.

Sometimes, the number of directly traffic density-reachable edges from the last edge in the hot route is larger than one. There are two causes for this. The first cause is multiple edges within one hot route. This can happen when *Eps* is larger than 0, and multiple edges of the hot route are in the *Eps*-neighborhood. This can be detected by checking to see if the *start*()'s and *end*()'s match across edges. In this case, only the *nearest* edge is appended to the hot route. The other edges will just be handled in the next iteration. The second cause is when a hot route *splits*. In this case, the current hot route is duplicated, and a different hot route is created for each split. The difference between these two cases can be detected by checking the directly traffic density-reachability condition between edges in the *Eps*-neighborhood.

The overall algorithm proceeds as follows. First, all hot route starts are extracted from the data. This is done by checking Definition 7 for every edge in G. This step has linear complexity because only individual edges with their *Eps*-neighborhoods are checked. Then, for every hot route start, the associated hot routes are extracted. Algorithm 1 shows a pseudo-code description.

One point of concern is efficiency. Suppose the adjacency matrix or list of the road network fits inside main memory. Then, searching the graph is quite efficient. Retrieving the list of TID's at each edge will require disk I/O but traversing the graph will not. However, suppose the adjacency matrix is too big to fit inside main memory. In this case, we introduce two additional indexing structures to help the search process. Figure 9 shows an illustration.

All vertices of the road network are stored in a 2D index, *e.g.*, R-tree (Vertex Index Tree). All edges are stored on disk (Edge Table). Each edge record consists of the edge ID and its starting and ending vertices (each vertex is an (x,

Algorithm 1. FlowScan

Input: Road network G, object trajectory data T, *Eps*, *MinTraffic*.
Output: Hot routes R

```
1: Initialize R to {}
2: Let H be the set of hot route starts in G according to T
3: for every hot route start h in H do
4:    r = new Hot_Route initialized to ⟨h⟩
5:    Add Extend_Hot_Routes(r) to R
6: end for
7: Return R
```

Procedure *Extend_Hot_Routes(hot route r)*

```
1: Let p be the last edge in r
2: Let Q be the set of directly traffic density-reachable neighbors of p
3: if Q is non-empty then
4:    for every split in Q do
5:       if route traffic density-reachable condition is satisfied then
6:          Let r' be a copy of r
7:          Append split's edges to r'
8:          Extend_Hot_Routes(r')
9:       end if
10:   end for
11: else
12:   Return r
13: end if
```

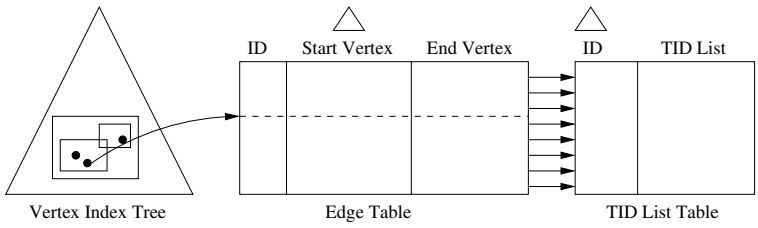


Fig. 9. Indexing structures of FlowScan

y) tuple). Using these two data structures, one can retrieve adjacent neighbors of an edge by querying the R-tree on the appropriate vertex and then retrieving the corresponding edges in the Edge Table. The R-tree is quite useful for finding adjacent neighbors of a specific edge since the coordinates of the adjacent neighbors tend to be close to that of the edge.

We note that the Edge Table may be accessed repeatedly to retrieve adjacent neighbors. This operation can be done more efficiently by exploiting locality. Specifically, if the physical locality of edges in the road network is preserved in the Edge Table, one can reduce the amount of disk I/Os. To this end, we create a *clustering* index on the Starting Vertex attribute of the Edge Table.

Assuming that each value is 4 bytes, each edge record is then 20 bytes. Then, if a page is 4K, it will contain approximately 200 edges. The intuition is that these 200 edges will be physically close to each other in the road network. Because FlowScan traverses through *Eps*-neighborhoods, it is highly likely that an edge and its neighbors will be stored on the same page in the Edge Table. In this case, disk I/O will be reduced because the page has already been fetched.

Lemma 2 (Completeness). *The set of hot routes discovered by* FlowScan *is complete and unique wrt. Eps and MinTraffic.*

Proof: The above assertion is easy to see because the construction algorithm uses the definition word-for-word, specifically Definition 6, to build the hot routes. Thus, given a hot route start, the set of hot routes extending from it is guaranteed to be found. The question is more about if the set of hot route starts found is complete. Because every edge in a hot route must satisfy the *MinTraffic* condition, there must be a "first" in a sequence. The set of hot route starts is simply these "firsts." Lastly, ordering is not a factor in FlowScan, because no marking or removal is done to G. Thus, it does not matter in which order H is processed. ■

4.4 Determining Parameters

There are two input parameters to the FlowScan algorithm: *Eps* and *MinTraffic*. The first parameter, *Eps*, controls how lax FlowScan can be between directly reachable edges. A value of 0 is too strict since it enforces strict spatial connectivity. A small value in the range of 2–5 is usually reasonable. In a metropolitan area, this corresponds to 2–5 city blocks; and in a rural area, this corresponds to 2–5 highway exists.

As for *MinTraffic*, this is often application or traffic dependent. "Dense" traffic in a city of 50,000 people is very different from "dense" traffic in a city of 5,000,000 people. In cases where domain knowledge dictates a threshold, that value can be used. If no domain knowledge is available, one can rely on statistical data to set *MinTraffic*. It has been shown that traffic density (and many other behaviors in nature) usually obeys the power law. That is, the vast majority of road segments have a small amount of traffic, and a relative small number have extremely high density. One can plot a frequency histogram of the edges and either visually pick a frequency as *MinTraffic* or use the parameters of the exponential equation to set *MinTraffic*.

5 Experiments

To show the effectiveness and efficiency of FlowScan, we test it against various datasets. FlowScan was implemented in C++ and all tests were performed on a Intel Core Duo 2 E6600 machine running Linux.

5.1 Data Generation

Due to the lack of real-world data, we used a network-based data generator provided by [1][3]. It uses a real-world city road network as the road network and generates moving objects on it. Objects are affected by the maximum speed on the road, the maximum capacity of the road, other objects on the road, routes, and other external factors.

The default generator provided generates essentially random traffic: an object's starting and end locations are randomly chosen within the network. In order to generate some interesting patterns, we modified how the generator chooses starting and end locations. Within a city network, "neighborhoods" are generated. Each neighborhood is generated by picking a random node and then expanding by a preset radius (3–5 edges). Moving objects are then restricted to start and end in neighborhoods.

Hot routes form naturally because of the moving object's preference for the quickest path. As a result, bigger roads (*e.g.*, highways) are more likely to be chosen by the moving objects. However, if too many objects take a highway or a road, it will reach capacity and actually slow down. In such cases, objects will choose to re-route and possibly create secondary hot routes.

5.2 Extraction Quality

General Results. To check the effectiveness of FlowScan, we test it against a variety of settings. First, we present the results for two general cases. Figure 10 shows several routes extracted from 10,000 objects moving in the San Francisco bay area. 10 neighborhoods of radius 3 each were placed randomly in the map. *Eps* and *MinTraffic* were set to 2 and 300, respectively. Each hot route is drawn in black with an arrow indicating the start and a dot indicating the end. The gray lines in the figures indicate all traffic observed in the input data (not the entire city map).

Even though the neighborhoods were completely random, we get realistic hot routes in this experiment. The hot routes in Figure 10(a) and 10(b) are CA-101 connecting San Francisco and San Jose, a major highway in the area. Figures 10(c) and 10(d) correspond to the Golden Gate Bridge connecting the city of San Francisco to the north. One of the random neighborhoods must have been across the bridge so objects had no choice but to use the bridge. Figure 10(e) shows a hot route connecting Oakland to that same neighborhood across the Richmond-San Rafael Bridge. Lastly, Figure 10(f) corresponds to a hot route connecting approximately Hayward to San Jose via I-880.

Next, Figures 11 shows three hot routes extracted from 5,000 objects moving in the San Joaquin network. Three neighborhoods were picked in this network, each with radius of 3. *Eps* and *MinTraffic* were set to 2 and 400, respectively. In Figures 11(a) and 11(b), the horizontal portions of the hot routes correspond to I-205. In Figure 11(b), the vertical portion corresponds to I-5. Both these roads are major interstate highways. The roads in Figure 11(c) are W. Linne Rd

[3] http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/

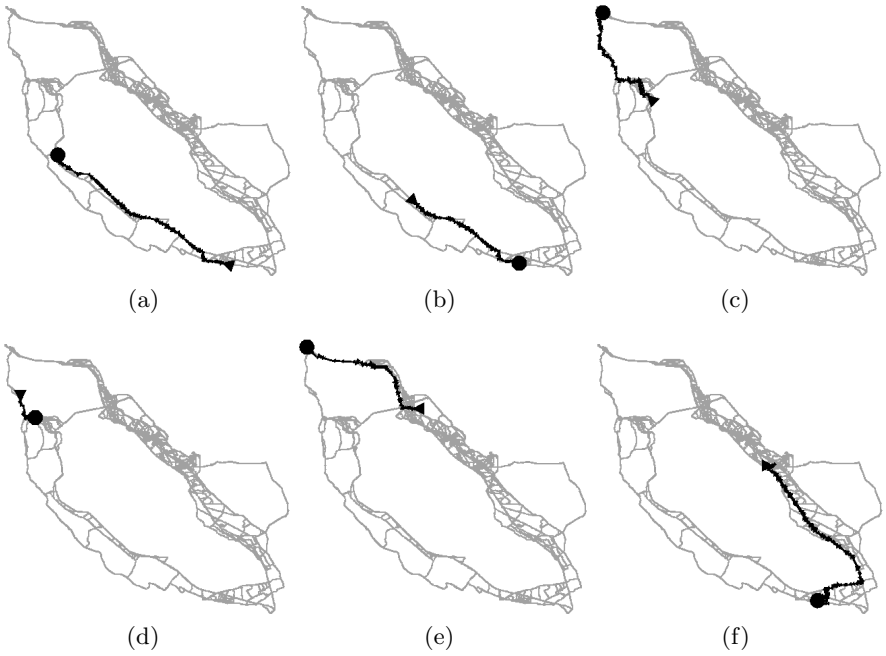


Fig. 10. Hot routes in San Francisco data map

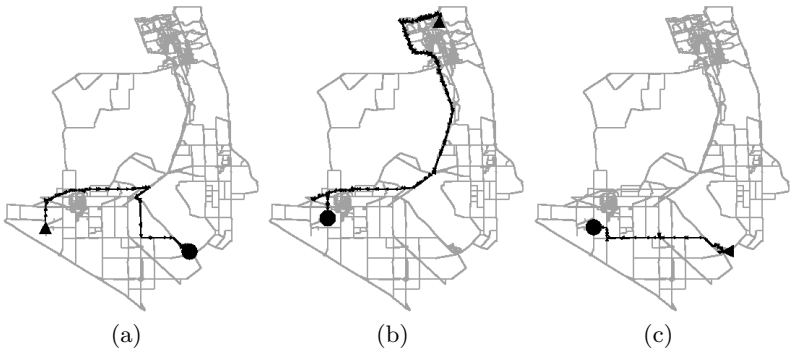


Fig. 11. Hot routes in San Joaquin data map

and Kason Rd. By looking at the city map, we observe that they make up the quickest route between the two neighborhoods.

Splitting Hot Route Behavior. We also tested FlowScan with some specific traffic behaviors. First, we test the case of a hot route splitting into two. This data set is generated by setting the number of neighborhoods in a road network to three and fixing the start node to be in one of the three. Because start and

destination neighborhoods cannot be the same, this forces the objects (1000 of them) to travel to one of two destinations. And because the objects like to travel on big roads (due to speed preference), they will usually leave the starting neighborhood using the same route regardless of the final destination and split sometime later.

Figures 12(a) and 12(b) shows the two hot routes extracted from the data. Both hot routes start at the green arrow at the lower right, move to the middle, and the split according to their final destinations. In Figure 12(c), the result from an AltGraph algorithm is shown. All edges that exceed the *MinTraffic* threshold (100) are connected if they are adjacent in the road network. Obviously, the two hot routes are connected together because the underlying objects are not considered. Figure 12(d) shows the result from a AltTrajectory algorithm [10]. In it, 14 clusters were found. Because shape is a major factor in trajectory clustering, the routes were broken into different clusters. The split is "detected" simply due to the hard left-turn shape, but the routes are not intact. One could post-process the results and merge near-by clusters, but this could run into the same problems as AltGraph since individual trajectories are ignored.

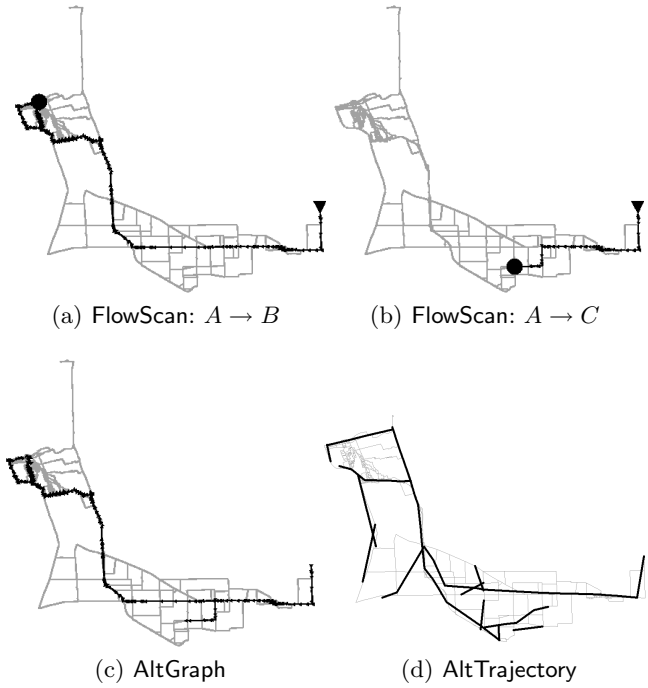
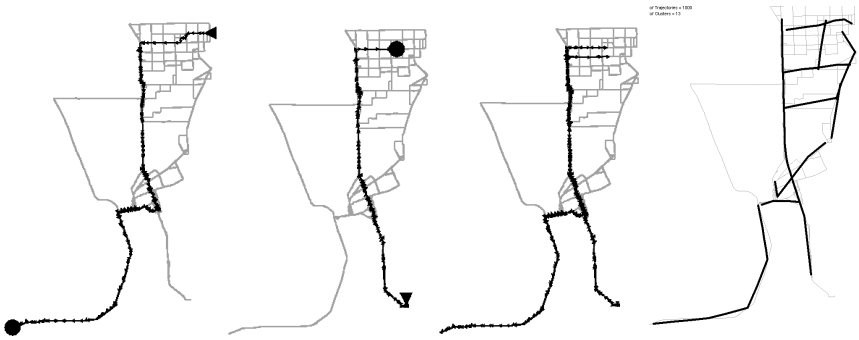


Fig. 12. Splitting hot routes

Overlapping Hot Route Behavior. Next, we test the case of two hot routes overlapping. That is, one starts at the same place as where the other one ends. To generate this data set, we also set the number of neighborhoods to three. Let

them be known as A, B, and C. Then, for half of the objects, their paths are $A \rightarrow B$; and for the other half, their paths are $B \rightarrow C$. We set the radius of neighborhood B to 0 to ensure that the two hot routes overlap.

Figure 13 shows the results of this test. As the graphs show, two hot routes were extracted. Figure 13(c) shows a result with an AltGraph algorithm. Although $B \rightarrow C$ (not shown) is correctly extracted in that algorithm, $A \rightarrow B$ is not. It is incorrectly linked together with $B \rightarrow C$ and erroneously forms $A \rightarrow B \rightarrow C$. This is because individual identities are not considered in the algorithm. Figure 13(d) shows the result of AltTrajectory. 13 clusters were discovered. Again, the routes are not intact. But more seriously, the trajectories near B are clustered into a single cluster because their shapes are similar.



(a) $B \rightarrow C$ (b) $A \rightarrow B$ (c) AltGraph (d) AltTrajectory

Fig. 13. Overlapping hot routes

5.3 Efficiency

Finally, we test the efficiency of FlowScan with respect to the number of objects. Figure 14 shows the running time as the number of objects increases from 2,000 to 10,000 with *MinTraffic* set to 10%. All objects were stored in memory, and time to read the input data is excluded. As the curve shows, running

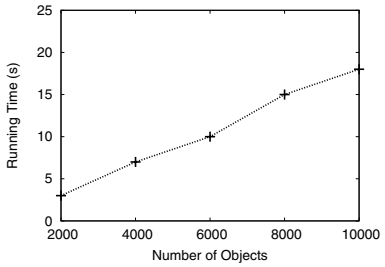


Fig. 14. Efficiency with respect to number of objects

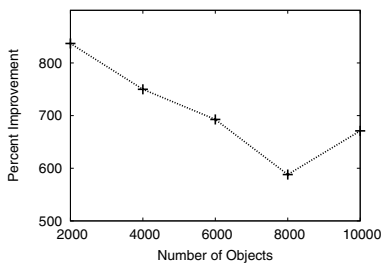


Fig. 15. Disk I/O improvement of clustered index on Edge Table

time increases linearly with respect to the number of objects. Next, we test the difference in disk I/O using a clustered Edge Table vs. an unclustered Edge Table. Figure 15 shows the result. Pages were set to 4K each and a buffer of 10 pages was used. We excluded the I/Os of the Vertex Index Tree and the TID List Table since they are the same in both cases. The figure shows the percent improvement of the clustered Edge Table. It is a significant improvement ranging from 588% to over 800%. This value is relatively stable because the percent improvement depends more on the structure of the network than the number of objects.

6 Related Work

Work in moving object clustering is closely related to FlowScan. Some examples include [6,2,8]. In [8], objects are grouped using a traditional clustering algorithm at each time snapshot and then linked together over time to form moving clusters. The assumption is that clusters will be stable across short time periods. In real world traffic, one can see how a traffic light or a incoming traffic from a highway on-ramp can easily breakup clusters between consecutive snapshots. As a result, [8] and similar approaches will likely find too many short clusters and miss the overall flow.

Work in the AltTrajectory class is also related. They include trajectory clustering [18,10] and trajectory modeling [11,9]. In both types, the focus is on individual objects, not aggregate. Further, most algorithms deal with free-moving objects and consider the shape in clustering. This is irrelevant in road networks. Also, the common traffic between sub-trajectories is ignored in the analysis. This could cause problems when hot routes merge or split or make drastic turns.

Data mining in spatiotemporal data is also related to our work. One class of problems mines sequential patterns of events (*e.g.*, temperature) at spatial locations [15]. Another problem is co-location mining [14,17,19]. A co-location rule states a set of locations that often occur together with respect to a neighborhood function. These work have a similar spirit of discovering frequent patterns, but they are different in that the input is not trajectory data. General data mining in sequential pattern mining [12] is another related area. Hot routes are similar to sequential patterns in trajectories. However, spatial information and traffic information are not considered in traditional data mining.

General moving object database research has work related to indexing [13,5,7] and similarity search [16,3]. But the focus of such work is on the raw edges, shapes, locations, etc. FlowScan focuses on a higher level problem.

Our definitions of density is similar in spirit to density-based clustering (DBSCAN [4]), also used in [10]. But the nature of the data is very different. A typical clustering algorithm is concerned with discovering clusters of points in spatial data, while FlowScan is concerned with discovering hot routes in traffic.

7 Conclusion and Future Work

In this study, we have examined the problem of discovering hot routes in road networks. Due to the complexity of the data, this is a problem not easily solved by existing algorithms in related areas. We show several typical traffic behaviors that are tricky to handle. To this end, we propose a new algorithm, FlowScan, which uses the density of traffic in sequences of road segments to discover hot routes. It is a robust algorithm that can handle the complexities in the data and we verify through extensive experiments. By comparing against other approaches, we see the advantages of this approach.

One important aspect of the trajectory data we did not utilize in FlowScan is the non-spatiotemporal information about the the trajectories. The type of the vehicle is one such example. The hot routes of sedans are sure to be different from the hot routes of transport trucks. Other attributes on the data facilitates a multi-dimensional approach to this problem. By knowing the correlations between hot routes and other attributes, one can enhance the usefulness of the discovered information.

References

1. Brinkhoff, T.: A framework for generating network-based moving objects. In: GeoInformatica'02 (2002)
2. Cadez, I.V., Gaffney, S., Smyth, P.: A general probabilistic framework for clustering individuals and objects. In: KDD'00 (2000)
3. Chen, L., Ozsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: SIGMOD'05 (2005)
4. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases. In: KDD'96 (1996)
5. Frentzos, E.: Indexing objects moving on fixed networks. In: Hadzilacos, T., Manolopoulos, Y., Roddick, J.F., Theodoridis, Y. (eds.) SSTD 2003. LNCS, vol. 2750, Springer, Heidelberg (2003)
6. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: KDD'99 (1999)
7. Güting, G.H., Schneider, M.: Moving Objects Databases. Morgan Kaufmann, San Francisco (2005)
8. Kalnis, P., Mamoulis, N., Bakiras, S.: On discovering moving clusters in spatiotemporal data. In: Bauzer Medeiros, C., Egenhofer, M.J., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, Springer, Heidelberg (2005)
9. Kostov, V., Ozawa, J., Yoshioka, M., Kudoh, T.: Travel destination prediction using frequent crossing pattern from driving history. In: ITSC'05 (2005)
10. Lee, J., Han, J., Whang, K.: Trajectory clustering: A partition-and-group framework. In: SIGMOD'07 (2007)
11. Liao, L., Fox, D., Kautz, H.: Learning and inferring transportation routines. In: AAAI'04 (2004)
12. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C.: Mining sequential patterns by pattern-growth: The prefixspan approach. TKDE'04 (2004)

13. Pfoser, D., Jensen, C.S.: Indexing of network constrained moving objects. In: GIS'03 (2003)
14. Shekhar, S., Huang, Y.: Discovering spatial co-location patterns: A summary of results. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, Springer, Heidelberg (2001)
15. Tsoukatos, I., Gunopulos, D.: Efficient mining of spatiotemporal patterns. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, Springer, Heidelberg (2001)
16. Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In: ICDE'02
17. Yoo, J.S., Shekhar, S.: A partial join approach to mining co-location patterns. In: GIS'04 (2004)
18. Zen, H., Tokuda, K., Kitamura, T.: A viterbi algorithm for a trajectory model derived from hmm with explicit relationship between static and dynamic features. In: ICASSP '04 (2004)
19. Zhang, X., Mamoulis, N., Cheung, D.W., Shou, Y.: Fast mining of spatial collocations. In: KDD'04 (2004)