Road-Network Aware Trajectory Clustering: Integrating Locality, Flow and Density

Binh Han, Ling Liu, Edward Omiecinski
College of Computing, Georgia Institute of Technology
{binhhan,lingliu,edwardo}@gatech.edu

Abstract—Mining trajectory data has been gaining significant interest in recent years. However, existing approaches to trajectory clustering are mainly based on density and Euclidean distance measures. We argue that when the utility of spatial clustering of mobile object trajectories is targeted at road-network aware location-based applications, density and Euclidean distance are no longer the effective measures. This is because traffic flows in a road network and the flow-based density characterization become important factors for finding interesting trajectory clusters. We propose NEAT−a road-network aware approach for fast and effective clustering of trajectories of mobile objects travelling in road networks. Our approach carefully considers the traffic *locality* characterized by the physical constraints of the road network, the traffic *flow* among consecutive road segments, and the flow-based *density* to organize trajectories into spatial clusters in a comprehensive three-phase clustering framework. NEAT discovers spatial clusters as groups of sub-trajectories which describe both dense and highly continuous flows of mobile objects. We perform extensive experiments with mobility traces generated using different scales of real road networks. Experimental results demonstrate the flexibility of the NEAT system and show that NEAT is highly accurate and runs orders of magnitude faster than existing density-based trajectory clustering approaches.

Index Terms—trajectory clustering; road network trajectory; location-base applications

1 Introduction

Recent years have witnessed a rapidly growth in the field of location-based services (LBSs) and applications due to the pervasive use of GPS receivers and WiFi or location sensing technology embedded in mobile devices (e.g. cellular phones, automobiles). With the number of smartphones in use world wide reaches 1.038 billion units in 2012 and is predicted to reach 2 billion units by 2015[1], LBS revenue is forecasted to reach an annual global total of \$13.5 billion by 2015[2], up from \$4 billion in 2012[3]. Ubiquitous GPS/WiFi-enabled mobile devices generate a huge amount of trajectory data, which are sequences of time-ordered locations of mobile objects. There has been a lot of work on collecting, storing, indexing and querying trajectories of mobile objects [1] [2] [3] [4] [5]. We refer to the trajectories of mobile objects in a road network as *MO trajectories*. Clustering trajectories of these objects provides the most value and has a wide range of LBS applications. For example, the resulting clusters would help provide knowledge about traffic flows as well as dense areas in a road network. Such knowledge is very useful for applications in vehicular ad hoc network (VANET) [6] [7], traffic monitoring [8], transportation planning [9] and location-based advertising [10]. We briefly present below two interesting application scenarios which show the usefulness of trajectory clustering and motivate us to study the problem of clustering trajectories of mobile objects moving in a spatially constrained road network.

Public transit planning. The establishment of public transportation system always target the road network routes which can maximize the utilization of public transportation vehicles. Since a bus runs along consecutive road segments, in order to make the best use of the bus service regarding the number of passengers it can serve and to improve public transport convenience, e.g. reduce the number of transit stops for passengers, knowing which routes with not only high density but also high continuity helps optimize rail/bus line and terminal arrangement.

Location based advertising on mobile devices. Mobile advertisers are trying to improve the matching of user locations and marketing information. It would be beneficial for local stores to send advertisements, e.g. special offers or discounts, to mobile devices taking path in major traffic flows passing by their stores. For example, consider when a store wants to send out text messages with a discount coupon. If the text message is sent to a group of people in a nearby dense road segment which is also part of a route with significant traffic flow leading to the store, it will better increase the chance that people receiving the coupon will actually come to the store during their trips, compared to when the message is sent to a dense area but does not belong to the traffic flow passing by their store.

A straight forward solution to address this clustering problem is to adapt the traditional density-based clustering algorithms (e.g. DBSCAN [11] or OPTICS [12] - a variant of DBSCAN) to group similar *MO* trajectories. However, clustering trajectories as a whole does not take into account similar sub-trajectories since trajectories have various lengths. This is addressed by partial trajectory clustering. The TraClus algorithm [13] is the representative method for clustering portions of a trajectory instead of the whole trajectory. Specifically, TraClus is a two phase clustering algorithm. In the partitioning phase, each trajectory is first examined sample by sample to identify a sequence of characteristic points at which the moving object makes a sharp turn, called a rapid change in direction, and then the trajectory is par-

1. http://www.strategyanalytics.com/
2. http://www.gartner.com/
3. http://www.abiresearch.com/

titioned into line segments by these characteristic points. The grouping phase performs a DBSCAN style clustering on those line segments to find similar sub-trajectories. A drawback to this and most similar partial trajectory clustering approaches is that they only consider distances in Euclidean space, while show reasonable performance for clustering trajectories of objects moving freely (e.g. the movement of animals through a forest or the movement of hurricanes across an ocean), are inappropriate for clustering *MO* trajectories. We argue that clustering *MO* trajectories should take into account the traffic locality characterized by the spatial constraints of the underlying network, e.g. road segments and intersections, the movement behavior of mobile objects as well as the traffic flows on consecutive road segments, in addition to mobile object density and the road network proximity. We illustrate our positional statements with the following set of examples.

Example 1. We have a set of four objects moving along the same road segment and produce four *MO* trajectories as shown in Figure 1(a). In the context of a road network, those objects have similar movement behavior with respect to the road segment. Therefore, their trajectories should be grouped together regardless of the difference in their specific movement on the road segment. Hence, it is unnecessary to further partition these trajectories, even though some rapid changes in direction are found in those trajectories.

Example 2. Consider two trajectories TR_1 and TR_2 in Figure 1(b). TR_1 describes an object moving on road segment RS_1, changing lanes, and then making a left turn to road segment RS_2. TR_2 describes another object moving straight from RS_1 to RS_3. If we use the TraClus algorithm, to cluster the *MO* trajectories, TR_1 will be partitioned into four line segments: A_1A_2, A_2A_3, A_3A_4, and A_4A_5, while TR_2 consists of only one line segment which is too long to be grouped with the segments of TR_1 on RS_1 (A_1A_2, A_2A_3, and A_3A_4). We argue that although there is no significant turning point found in TR_2, we should still partition it into two line segments fragments corresponding to the two distinct road segments RS_1 and RS_3.

Example 3. In Figure 1(c), road segments RS_1 and RS_3 share a larger number of mobile objects than that of road segments RS_1 and RS_2. Thus, when consider traffic continuity, traffic on road segment RS_1 should be merged with traffic on RS_3 rather than RS_2.

Example 4. We have three trajectories: TR_1 and TR_2 are on the same road segment, and TR_3 is on another road segment as shown in Figure 1(d). Using the Euclidean distance, TR_3 is closer to TR_2 than TR_1, even though using the road network distance (either segment length based or travel time based), TR_1 is closer to TR_2 than TR_3.

In this paper, we present NEAT−a road NEtwork Aware approach to Trajectory clustering. To the best of our knowledge, NEAT is the first technique to address the *MO* trajectory clustering problem by taking into account the continuity of movements restricted by the underlying road network, the network proximity and the traffic flows among consecutive road segments to organize *MO* trajectories into spatial clusters. The clusters discovered by NEAT are groups of sub-trajectories, which describe both dense and highly continuous traffic flows of mobile objects. A unique feature of the NEAT framework is the formulation of the three phase trajectory partitioning, merging and clustering process. Specifically, based on the observations from the above examples, we identify three important design guidelines. First, for a given set of *MO* trajectories, the road intersections can be viewed as the initial partitioning points where trajectories can be split into atomic sub-trajectories called trajectory fragments. Second, the trajectory fragments corresponding to a road segment can be viewed as a locally dense cluster of objects involved in the given set of trajectories. Third, trajectory fragments should be clustered based on their continuity with regard to the traffic flows on the consecutive road segments. In addition, the proximity measure in a road network space uses shortest path distance instead of Euclidean distance. We perform extensive experiments with road network mobility traces generated using different scales of road network maps. Our experimental results demonstrate that the NEAT approach is highly efficient and accurate. It can run more than three orders of magnitude faster than existing density-based trajectory clustering approaches.

2 NEAT MODEL AND FRAMEWORK

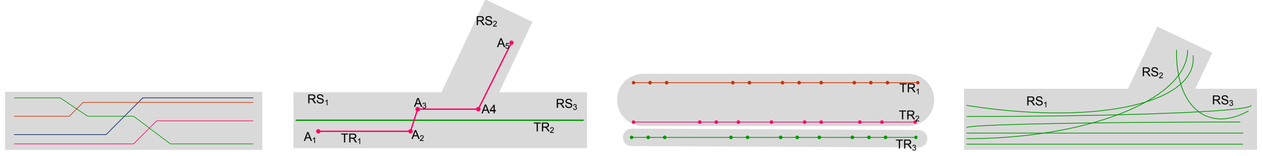
We first describe a reference model for road networks and present the basic concepts and operations of the NEAT model. We end this section with a brief overview of the NEAT three phase framework and an illustrative example.

2.1 Road-Network Model

A road network is represented by a single directed graph $G = (\mathcal{V}, \mathcal{E})$, composed of the junction nodes $\mathcal{V} = \{n_0, n_1, \ldots, n_N\}$ and directed edges $\mathcal{E} = \{(sid, n_in_j) | n_i, n_j \in \mathcal{V}\}$.

An edge $e = (sid, n_in_j) \in \mathcal{E}$ representing a road segment connecting two junctions n_i and n_j in the real road network. The listed order n_in_j indicates the direction from n_i to n_j of the road segment. For road segments which have bidirectional lanes, we use edge $e = (sid, n_in_j)$ and $e' = (sid, n_jn_i)$ to denote the fact that the road segment is bi-directional and we label each edge with the corresponding road segment identifier sid. The length of a road segment $e = (sid, n_in_j)$ is denoted by $length(n_in_j)$.

Let $L(e)$ denote the set of adjacent edges of $e = (sid, n_in_j)$ and $L_{n_i}(e)$ denote the set of adjacent edges of e, which connect to e at junction n_i. Hence, we have $L(e) = L_{n_i}(e) \cup L_{n_j}(e)$. If n_i is a dead-end node connected by edge e, then $L_{n_i}(e) = \phi$. If two edges e_i and e_j are adjacent, function $I(e_i, e_j)$ will return the junction node (intersection) of these two edges. A route in the road network G is a network path $e_0e_1...e_k$ such that $e_{i+1} \in L(e_i)$ $(0 \leq i < k)$.



(a) An example of the movement of four vehicles on a road segment

(b) Parts of two trajectories on the same road segment that are overlooked in TraClus

(c) Traffic on road segment RS_1 should be merged with traffic on road segment RS_3 rather than RS_2

(d) An example showing the difference between the Euclidean proximity and the road network proximity

Fig. 1: Illustrative examples

We define a *road network location* of a mobile object as a tuple of three elements: *sid* $-$ the identifier of the road segment on which this object resides, the geometric coordinates (x, y) of the position of the object on the road segment *sid*, and the timestamp t when the position is recorded, denoted by $l = (sid, x, y, t)$. A road network location can also be represented by a tuple (sid, p, t) where p is the offset of the location from the start junction of the road segment identified by sid. We use the (x, y) coordinates to represent location in this paper due to the popularity of geometric coordinates. We measure the network proximity between two network locations using the network-based distance metric [14]. Let l_i and l_j denote the network locations that belong to the edge $e_i = (sid_i, n_a n_b)$ and the edge $e_j = (sid_j, n_c n_d)$ respectively and $e_i \neq e_j$. The network distance between l_i and l_j is the length of the shortest path between l_i and l_j, denoted by $dist_N(l_i, l_j)$. We use the terms point and location interchangeably to refer to a road network location and the terms junction, intersection and endpoint interchangeably to refer to a road junction. We use $(sid, n_i n_j)$ and $n_i n_j$ interchangeably to denote a segment connecting two end nodes n_i and n_j when there is no confusion.

2.2 NEAT model

In this section, we define the basic concepts and operations of our road network aware trajectory model with illustrative examples.

For each mobile object, each of his/her trips with a beginning location and a destination location forms a trajectory. A *trajectory*, denoted by $TR = (trid, l_0 l_1 ... l_n)$, is a time-ordered sequence of locations $l_0, l_1, ..., l_n$ of an *MO* in the road network over time and uniquely identified by a trajectory identifier *trid*. A subsequence of points in a trajectory forms a *subtrajectory*. Note that in our model, the temporal information in a trajectory, i.e. the recorded timestamps, determines the order of locations in the trajectory. Therefore, the direction of movement of an object is always preserved. For presentation convenience, we do not explicitly mention the directions of movement in the definitions and figures used in the subsequent sections of the paper when no confusion occurs.

Definition 1. *(t-fragment)* Let $TR = \{trid, l_0 l_1 ... l_n\}$ denote a trajectory consisting of $n+1$ points and *trid* denote the trajectory identifier. A t-fragment of TR, denoted by $tf = \{trid, sid, l_k l_{k+m}\}$, represents a subtrajectory $l_k l_{k+1} ... l_{k+m}$ consisting of $m+1$ consecutive points extracted from TR which lie on the same road segment *sid*, i.e. $l_i.sid = l_j.sid$ $(\forall i, j : k \leq i, j \leq k+m, i \neq j)$.

There can be more than one t-fragment associated with a road segment for a given trajectory (in case the mobile object travels on the road segment multiple times during her trip).

Definition 2. *(base cluster)* Let $\mathcal{T}$ denote a set of trajectories and e denote a road segment. A *base cluster* S with respect to e is a group of distinct t-fragments, each of these t-fragments belongs to a trajectory in $\mathcal{T}$ and is associated with e. The *base cluster* S is formally defined as follows:

$S = \{tf_i | TR(tf_i) \in \mathcal{T}, tf_i.sid = e.sid\}$ where $TR(tf_i)$ denotes the trajectory from which the t-fragment $tf_i \in S$ is extracted. The road segment e is called the representative of the base cluster S, and is denoted by e^S. The base cluster S is said to be associated with the road segment e^S.

We call a trajectory which has t-fragments in a base cluster the *participating trajectory* of the base cluster.

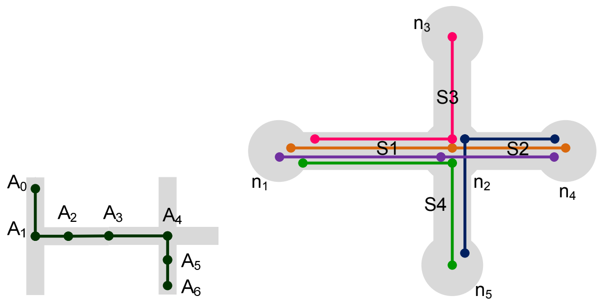
Definition 3. *(trajectory cardinality)* The set of participating trajectories of a base cluster S is defined as: $PTr(S) = \{TR(tf_i) | \forall tf_i \in S\}$. The cardinality of $PTr(S)$, denoted by $|PTr(S)|$, is called the *trajectory cardinality* of S.

Definition 4. *(cluster density)* The *density of a base cluster* S, denoted by $d(S)$, is the number of t-fragments in S. Given $B = \{S_0, S_1, ..., S_N\}$ is a set of base clusters, we call the base cluster with the highest density in B the *dense-core* of B, denoted by $densecore(B)$.

Definition 5. *(netflow)* The *netflow* between two base clusters S_i and S_j, denoted by $f(S_i, S_j)$, is the number of trajectories participating in both clusters: $f(S_i, S_j) = |PTr(S_i) \cap PTr(S_j)|$

The function *netflow* between two base clusters computes the number of common objects traveled on both representative road segments e^{S_i} and e^{S_j}.

Definition 6. *(f-neighborhood)* Let B denote a set of base clusters, S_i denote a base cluster and n_u denote one endpoint of e^{S_i}. The f-neighborhood of S_i *wrt.* n_u, denoted by $N_f(S_i, n_u)$, is the set of base clusters that have at least one common participating trajectory, and is formally defined as: $N_f(S_i, n_u) = \{S_j | \; e^{S_j} \in$



(a) A trajectory has three t-fragments (b) An example of base clusters and flow cluster

Fig. 2: Examples of the elements in the NEAT model

$L_{n_u}(e^{S_i})$ & $f(S_i, S_j) > 0\}$.

Let n_v be the other endpoint of e^{S_i}. We define the f-neighborhood of S_i *wrt.* e^{S_i} as: $N_f(S_i) = N_f(S_i, n_u) \cup N_f(S_i, n_v)$. Each $S_j \in N_f(S_i)$ is called the f-neighbor of S_i. Note that the f-neighbor is a symmetric relation.

Definition 7. *($maxFlow$-neighbor)* Let S_i denote a base cluster and n_u denote one endpoint of e^{S_i}. We call S_k the $maxFlow$-neighbor of S_i at n_u, denoted by $maxFlow(S_i, n_u)$, if $f(S_i, S_k) = max\{f(S_i, S_j)|\ S_j \in N_f(S_i, n_u)\}$. $f(S_i, S_k)$ is called a $maxFlow$ of S_i.

Definition 8. *(flow cluster)* A *flow cluster* (or a *flow* for short) is an ordered list of base clusters, denoted by $F = \{S_0, S_1, ..., S_N\}$, where $S_{i+1} \in N_f(S_i) (0 \leq i < N)$ and $e^{S_0}e^{S_1}...e^{S_N}$ forms a route in the road network. We call $e^{S_0}e^{S_1}...e^{S_N}$ the representative route of F, denoted by r_F.

Since a base cluster is comprised of t-fragments and a flow cluster is comprised of base clusters, we say that a flow cluster is comprised of t-fragments. Therefore the definition of *trajectory cardinality* also applies to a flow cluster. We define the netflow between a flow cluster F and a base cluster S as $f(F, S) = |PTr(F) \cap PTr(S)|$.

Figure 2(a) shows a trajectory which can be represented by a sequence of three t-fragments A_0A_1, A_1A_4 and A_4A_6. In Figure 2(b), we have five trajectories located on four road segments n_1n_2, n_2n_3, n_2n_4 and n_2n_5. There are three t-fragments which lie on n_1n_2. Those three t-fragments are grouped together in base cluster S_1 whose representative road segment is n_1n_2. In total, we have a set of base clusters $B = \{S_1, S_2, S_3, S_4\}$. The density of S_1 is $d(S_1) = 4$. Similarly, we have $d(S_2) = 3$, $d(S_3) = 1$ and $d(S_4) = 2$. S_1 is the dense-core of B since $d(S_1) = 4$ is the highest density. The netflows among these base clusters are: $f(S_1, S_2) = 2$, $f(S_1, S_3) = 1$, $f(S_1, S_4) = 1$, $f(S_2, S_3) = 0$ and $f(S_2, S_4) = 1$. The f-neighborhood of S_1 *wrt.* n_2 is $N_f(S_1, n_2) = \{S_2, S_3, S_4\}$, in which S_2 is the $maxFlow$-neighbor of S_1. The possible flow clusters include $\{S_1, S_2\}$, $\{S_1, S_3\}$, $\{S_1, S_4\}$ and $\{S_2, S_4\}$.

2.3 NEAT Framework Overview

Given a road network $G = (\mathcal{V}, \mathcal{E})$ and a set of N trajectories collected from mobile objects travelling on G, denoted by $\mathcal{T} = \{TR_1, TR_2, ..., TR_N\}$, NEAT performs road network aware trajectory clustering in three phases:

Phase 1 - Base cluster formation: We transform the given set of *MO* trajectories into a set of trajectory fragments (t-fragments). Then we organize these t-fragments into *base clusters* by grouping those t-fragments that correspond to the same road segment into one base cluster.

Phase 2 - Flow cluster formation: We selectively merge *base clusters* into *flow clusters* based on the major mobility flows and the flow continuity inherent in the given set of trajectories.

Phase 3 - Flow cluster refinement: We optimize the clustering result using a density-based refinement method. Our density-based optimization modifies the widely-used Hausdorff distance with the shortest path measurement and adapts the DBSCAN clustering algorithm [11].

The final result produced by NEAT is a partitioning of the given *MO* trajectories into a set of trajectory clusters $\mathcal{O} = \{C_1, C_2, ..., C_K\}$ where each cluster $C_i (0 \leq i \leq K)$ contains a set of trajectory fragments satisfying two criteria: (1) *High density*: the trajectory fragments in the same cluster are within the network proximity of each other; (2) *High continuity*: the trajectory fragments in the same cluster show a major traffic flow in the given trajectory data.

The NEAT system uses 3-tier client/server architecture. Each client node acts as a mobile device which records its locations, sends its trajectories to a NEAT server and makes requests to the server to get trajectory clustering results for a particular road network. NEAT server also distributes trajectory datasets across multiple nodes in a cluster. These data nodes can perform some data preprocessing tasks. In this paper, we focus on the trajectory clustering application running on the NEAT server.

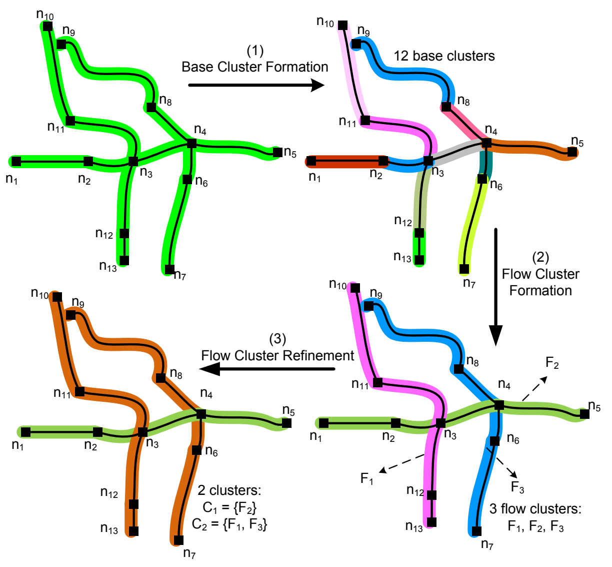


Fig. 3: An example of three phase clustering in the NEAT framework.

Figure 3 illustrates the three phase NEAT framework. Consider a set of trajectories located on 12 road segments in an example road network as shown in the upper-left of Figure 3. In Phase 1, a set of base clusters are constructed from the given set of trajectories (the upper-right of Figure 3). In Phase 2, we utilize road network information and mobile object movement characteristics to group base clusters into flow clusters. Important operations include computing the *netflow*s, finding the f-neighborhoods and compute the $maxFlow$-neighbors. For our example, the result of this phase is a set of three flow clusters $\{F_1, F_2, F_3\}$ (the bottom-right of Figure 3). In Phase 3, we refine the resulting flow clusters by merging those flow clusters that are close in terms of a network based distance measure and a distance threshold. In Figure 3, F_1 and F_3 are merged in Phase 3 to form a larger trajectory cluster. The two clusters C_1 and C_2 shown in the bottom-left of Figure 3 are the final result of clustering for this specific example.

3 THREE PHASE TRAJECTORY CLUSTERING

In this section, we present in details the algorithms used in the NEAT clustering framework to produce base clusters in Phase 1, flow clusters in Phase 2 and the final trajectory clusters in Phase 3.

3.1 Phase 1 - Base Cluster Formation

We perform the base cluster formation in two steps. First, we examine the input set of *MO* trajectories and partition each *MO* trajectory into a sequence of t-fragment. Second, we group those t-fragments that belong to the same road segments into one *base cluster*.

3.1.1 *Partitioning Trajectories into t-fragments*

Since a mobile object moves along contiguous road segments, two consecutive locations recorded in a *MO* trajectory are either on the same road segment or on two different road segments. In the latter case, the two different road segments are either contiguous or lie on the route (travel path) of the mobile object such that they are connected through a sequence of road junction nodes on the same path. For each trajectory $TR_k = \{trid_k, l_0 l_1 ... l_n\}$ in the given trajectory dataset, we start the t-fragment extraction by examining TR_k from the first location l_0 to the last location l_n in the sequence of location samples $\{l_0 l_1 ... l_n\}$ of the trajectory. Based on the fact that a mobile object has to go through the road intersection when moving between two contiguous road segments, we take every two consecutive points in the trajectory, say l_i and l_{i+1}, and check if their road segment identifiers, denoted by $sid(l_i)$ and $sid(l_{i+1})$ respectively, are different. If $sid(l_i) \neq sid(l_{i+1})$, we know that l_i and l_{i+1} are on different road segments. If they are contiguous, we can obtain the road junction node that intersects these two road segments. If the two road segments happen to be not contiguous, we can obtain the sequence of road junction nodes connecting them on the travel path of the object using the map-matching approach [15]. Next, we insert the obtained junction node(s) as new points in between l_i and l_{i+1} in the trajectory being examined. The junction nodes added to a trajectory in this phase are marked as different points than the original location samples. After examining every point in a given trajectory TR_k, the sequence of junction nodes added to TR_k will serve as the trajectory splitting points used to partition the trajectory into t-fragments. When a given set of trajectories are given as time series of geometric coordinates, NEAT will first preprocess the set of trajectories using map-matching (MM) algorithms such that each point in a trajectory is mapped to a road network location as defined in Section 2.1. We use the SLAMM map-matching algorithm [15] in this data prepocessing step. MM algorithms for bulk location data are more effective as noted in [15] because look-ahead and look-around algorithms can catch many known errors of earlier MM algorithms, such as map-matching location samples between two nearby parallel road segments.

By transforming a trajectory into a set of t-fragments, only the first point and the last point in the original trajectory are kept, together with the newly inserted road junction points. These points play critical roles in extracting t-fragments and constructing base clusters in the next step of Phase 1 as well as in subsequent phases of NEAT. Furthermore, the sequence of t-fragments extracted from a trajectory still maintains the traveling route, the movement direction as well as the identifier of the original trajectory.

3.1.2 *Grouping t-fragments into Base Clusters*

We examine the t-fragments extracted from the *MO* trajectories and group them by their road segment identifiers. Each group of t-fragments corresponding to a road segment forms one base cluster with the road segment as its representative (Definition 2). As discussed in Section 1, the t-fragments on the same road segment are considered close in terms of network proximity and they display similarity in the movement of their mobile objects. We compute the density of the resulting base clusters (Definition 4), then sort them by their densities in descending order. The output of Phase 1 is a sorted list of base clusters with the first base cluster as the dense-core of the set of base clusters. The base clusters are used as the building blocks of our flow-based clustering in the next phase. We will use flow and density controlled merging algorithms to construct the final trajectory clusters of a given trajectory dataset $\mathcal{T}$.

3.2 Phase 2 - Flow Cluster Formation

The flow-based clustering algorithm takes as an input the list of base clusters B produced from Phase 1. It starts by selecting one base cluster in B as the first initial flow cluster. It then expands this initial cluster by adding other base clusters one at a time such that the representative road segments of the base clusters selected for merging are concatenated to make a route. This expanding process will stop when every base cluster in B has been examined for its potential to be merged with existing flow clusters. We consider flow, density and road speed limit as three characteristics of a traffic stream to define a set of merging criteria. We construct flow clusters by grouping base clusters according to these criteria. We discuss below how to choose the initial base cluster and determine which base clusters are the best candidates for merging with the flow cluster under consideration.

3.2.1 Density-based Flow Cluster Initialization

In the first prototype of NEAT, we take the densecore of the density-ordered list of base clusters B to begin the flow-based clustering process. There are at least two reasons for choosing $densecore(B)$ as the initial flow cluster to start Phase 2. Given that a major traffic stream usually covers the road segment(s) with the highest traffic density in the road network, if we randomly pick a base cluster in B to initialize a flow cluster, it might lead to a flow cluster that describes a negligible traffic stream and will eventually be filtered out. In addition, choosing $densecore(B)$ also guarantees that the set of base clusters are merged in a deterministic order. Hence, the resulting flow clusters are always the same for the same input set of trajectories.

3.2.2 f-neighbor Merging for Flow Clusters

In this section, we describe how to merge a base cluster into an existing flow cluster. Suppose we are in the process of expanding a flow cluster F, which is in the form of an ordered list of base clusters, denoted by $\{S_a S_{a+1} \ldots S_{b-1} S_b\}$. We extend the list by inserting a base cluster either at the front or at the end of the list. Inserting a base cluster at the front of the list is performed similarly to inserting a base cluster at the end of the list. Let n_u denote the other end point of e^{S_b} other than the intersection $I(e^{S_{b-1}}, e^{S_b})$ of the two representative road segments of base clusters S_{b-1} and S_b. The base cluster S_{b+1} to be added to the end of the list has to be an f-neighbor of S_b *wrt.* n_u, i.e. $S_{b+1} \in N_f(S_b, n_u)$. As it is well known that flow, density and velocity are the three variables of traditional theory for uninterrupted traffic flow [16], we choose S_{b+1} among the base clusters in $N_f(S_b, n_u)$ by considering the *flow factor* q, *density factor* k and *speed limit factor* v.

Definition 9. Given a base cluster S and n_u as one endpoint of road segment e^S, the *flow factor* q, *density factor* k and *speed limit factor* v of a base cluster $S_j \in N_f(S, n_u)$ *wrt.* S are defined respectively as follows:

$$q = f\left(S, S_j\right) / |PTr(S)| \tag{1}$$

$$k = d\left(S_j\right) / (d(S) + \sum_{S_i \in N_f(S, n_u)} d\left(S_i\right)) \tag{2}$$

$$v = speed\left(S_j\right) / \sum_{S_i \in N_f(S, n_u)} speed\left(S_i\right) \tag{3}$$

where $speed(S_j)$ is the speed limit of e^{S_j}.

Definition 10. Given a base cluster S and n_u as one endpoint of e^S, the *merging selectivity* of a base cluster $S_j \in N_f(S, n_u)$ is defined as:

$$SF(S, S_j) = w_q \times q + w_k \times k + w_v \times v \tag{4}$$

where the coefficients w_q, w_k and w_v determine the weights of q, k, v respectively. The weights $w_q \geq 0$, $w_k \geq 0$ and $w_v \geq 0$ satisfy $w_q + w_k + w_v = 1$.

Here we assume that S is currently either the first element or the last element of a flow cluster F. Thus, selecting a base cluster to merge with F implies selecting a base cluster to merge with S. The three factors q, k, v are computed for each candidate base cluster S_j and normalized between [0,1] to better represent their relative capability of extending the flow cluster under examination to form a route with high continuity and density as our clustering objectives. Specifically, the flow factor q measures the relative netflow between S and S_j, thus, high q means that S_j maintains high continuity of the traffic flow when merging with S. The density factor k measures the relative density of S_j among neighboring base clusters at the road junction shared with S, namely n_u, which contributes to the density characteristic of the extended flow. The speed limit factor v measures how fast mobile objects can move within S_j compared to its neighbors at n_u, which is also a capable metric to find popular routes as mobile objects tend to travel following routes with either shortest distances or shortest travel times. In Definition 10, we combine all three factors q, k, v under a weighing scheme to select the most relevant base cluster to merge wrt. our clustering objectives. According to the above definitions, the base cluster in $N_f(S, n_u)$ which has the highest merging selectivity, denoted by SF_{max}, will be chosen to merge with S.

Each base cluster which has been merged into a flow cluster is removed from B. We also use a threshold *minCard* for the trajectory cardinality of a flow cluster to filter out those flow clusters whose trajectory cardinalities are smaller than *minCard*. Finally, the condition to stop expanding the list $\{S_a S_{a+1} \ldots S_{b-1} S_b\}$ at the end of the list is when S_b has no f-neighbor *wrt.* its endpoint n_u, i.e. $N_f(S_b, n_u) = \emptyset$. A similar condition is applied to stop expanding the list at the front. When both conditions are reached, we add the resulting flow cluster to $\mathcal{W}$, the output set of flow clusters in Phase 2. Then, we begin the next iteration of flow-based clustering with the remaining base clusters in the list B with the same process as described above, until all the base clusters are assigned to flow clusters, i.e. B becomes empty.

3.2.3 Decisions on f-neighbor Merging

Suppose we are at a merging step and the highest merging selectivity at this step is SF_{max}. We discuss the case when there are more than one base cluster which have the merging selectivity values of SF_{max} at a merging step. Suppose a base cluster S is at one end of an intermediate flow cluster F and S has m equally qualified neighbors $S_1, S_2, \ldots, S_m$ to merge wrt. their merging selectivity values, i.e. $SF(S, S_1) = \ldots SF(S, S_m) = SF_{max}$. In our *flow-based clustering algorithm* described above, we randomly pick one base cluster in $\{S_1, S_2, \ldots, S_m\}$. We call it "random SF_{max}" merging scheme. Here we propose two schemes in which instead of randomly picking, we carefully consider the potential merging opportunity of each equally qualified base cluster, called "look-back" and "look-ahead" schemes respectively. Both schemes focus on the flow property of the traffic stream.

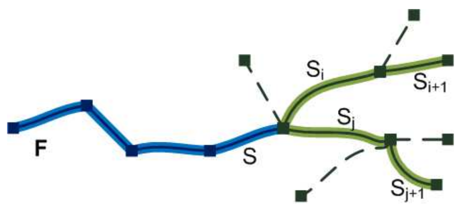


Fig. 4: An example of a merging iteration in which a base cluster S at the end of an intermediate flow cluster F has two neighbors with $SF(S, S_i) = SF(S, S_j) = SF_{max}$

Look-back Merging. We compare the netflows between the flow cluster under examination and m base clusters $f(F, S_i), f(F, S_2), ..., f(F, S_m)$ and choose one $S_j (1 \leq j \leq m)$ such that $f(F, S_j) = max\{f(F, S_1), f(F, S_2), ..., f(F, S_m)\}$. This means we want to maintain the movement of as many objects which have already on the representative route of the flow cluster F as possible. So that we "look-back" to the flow cluster F which has been extended so far and pick among those m candidates the one that share the largest number of participating trajectories with F.

Look-ahead Merging. Let n_u be the intersection of e^S and e^{S_1}, e^{S_2},..., e^{S_m}. We consider the $maxFlows$ of those m candidates at the other endpoints, which are not n_u, of their representative road segments. If the $maxFlow$ of S_j has the biggest value among them, we will select S_j to merge with S. In this scheme, we "look-ahead" to the $maxFlow$ at the other endpoint of each candidate's representative road segment and select the largest one. By using "look-ahead" scheme, we want to make sure that we capture all the significant netflows while expanding a flow cluster.

An example of this case is illustrated in Figure 4 where a base cluster S at the end of an intermediate flow cluster F with $SF(S, S_i) = SF(S, S_j) = SF_{max}$. In the "random SF_{max}" scheme, S_i and S_j have equal merging opportunity, thus, we randomly pick either S_i or S_j to merge with S. In the "look-back" scheme, we pick the base cluster which gives $max(f(F, S_i), f(F, S_i))$. In the "look-ahead" scheme, let S_{i+1} be a $maxFlow - neighbor$ of S_i and S_{j+1} be a $maxFlow - neighbor$ of S_j, we pick the base cluster either S_i or S_j based on which one gives $max(f(S_i, S_{i+1}), f(S_j, S_{j+1}))$.

3.2.4 Weight Assignment Criteria

The setting of the weights w_q, w_k and w_v is usually determined by the specific location-based applications. If an application favors the three factors equally when considering a traffic stream, we can set $w_q = w_k = w_v = 1/3$. If we set (w_q, w_k, w_v) = (0, 1, 0), we will merge a base cluster with its densest f-neighbor. The resulting flows in this case will describe a route where traffic is highly concentrated. A combination of (w_q, w_k, w_v) = (0, 0, 1) will produce flow clusters that describe the routes where objects can travel the fastest. For traffic monitoring applications, the flow factor and the density factor can be considered the most important factors so that we can set (w_q, w_k, w_v) to (1/2, 1/2, 0). If the application emphasizes the flow property of a traffic stream and considers the flow factor as the most important one, it can set $w_q = 1$ and $w_k = w_v = 0$. Therefore, the $maxFlow$-neighbor of S (Definition 7) will be selected to merge with S.

Beside user supplied weights, NEAT also supports system generated weights to make it run in an automated manner. For system supplied weights, we introduce an *adaptive weight assignment* scheme which selects the most critical characteristics of the traffic stream based on the values of the flow, density and speed limit factors at each merging step. Suppose we are in the process of forming a flow cluster. At each merging step, we compute the maximum values q_{max}, k_{max}, v_{max} of the flow factor q, density factor k and speed limit factor v respectively. We the assign the weights w_q, w_k and w_v to be used at the corresponding merging step as follows:

$$w_q = q_{max}/(q_{max} + k_{max} + v_{max}) \tag{5}$$

$$w_k = k_{max}/(q_{max} + k_{max} + v_{max}) \tag{6}$$

$$w_v = v_{max}/(q_{max} + k_{max} + v_{max}) \tag{7}$$

Note that all three factors q, k, v are normalized between 0 and 1. Thus, their maximum values at each merging step q_{max}, k_{max}, v_{max} can show the relative importance of each property of the traffic stream passing the road intersection corresponding to the base cluster under consideration in the merging step. By adaptively assigning weights based on those maximum values as in Formula 5, 6, 7, NEAT puts more weight on the critical factor at each merging step and therefore, automatically discovers flow clusters which capture important traffic flow inherent from the given trajectory data.

The pseudo code of flow-based clustering is presented in Algorithm 1. It performs on a set of base clusters B until all the base clusters are assigned to flow clusters, i.e. B is empty (lines 3-11). Each round of flow-based clustering starts with the dense core of B (line 4). The procedure $expandFlow()$ (lines 19-32) performs f-neighbor merging to extend a flow from its both sides (lines 8-9) given the factor weights w_q, w_k, w_v. If the input weights (w_q, w_k, w_v) are set to $(0, 0, 0)$, the adaptive weight assignment scheme is used to adaptively compute the weights at each merging step (line 21). It computes the f-neighborhood of the base cluster at one end of the flow (line 19) then select the f-neighbor with maximum merging selectivity to assign to the current flow. The flow continues to be expanded (line 31) as long as there are candidates to be merged (line 23), otherwise it stops. We also filter out the flows with insufficient number of participating trajectories. We set $minCard$ to equal to the average trajectory cardinality of the flow clusters in the set W. Those flow clusters whose trajectory cardinalities are smaller than $minCard$ will be discarded (lines 13-17).

Algorithm 1 Flow-based clustering

```
Input: (1) A set of base clusters $B = \{S_0, S_1, ..., S_M\}$
       (2) A road network $G$
       (3) Factor weights $w_q$, $w_k$, $w_v$
Output: A set of flow clusters $W = \{F_1, F_2, ..., F_Q\}$
1: $flowId = 0$;
2: $W = \emptyset$;
3: while $B \neq \emptyset$ do
4:     $S_c = densecore(B)$;
5:     add $S_c$ to $F_{flowId}$;
6:     remove $S_c$ from $B$;
7:     $\{n_{c1}, n_{c2}\}$ =getEndPoints($e^{S_c}$);
8:     expandFlow($S_c, n_{c1}, flowId, w_q, w_k, w_v$);
9:     expandFlow($S_c, n_{c2}, flowId, w_q, w_k, w_v$);
10:    $flowId$++;
11: end while
12: $minCard$ =averagePtr($W$);
13: for each $F_i \in W$ do
14:     if Ptr($F_i$) $< minCard$ then
15:         remove $F_i$ from $W$
16:     end if
17: end for
18: Procedure expandFlow($S, n_u, flowId, w_q, w_k, w_v$){
19: $N_f$ =getfNeigborhood($S, n_u$);
20: if $(w_q, w_k, w_v) = (0, 0, 0)$ then
21:     $(w_q, w_k, w_v) = getAdaptiveWeights(S, N_f)$;
22: end if
23: if $N_f \neq \emptyset$ then
24:     for each $S_i \in$N$_f$ do
25:         $SF(S_i) =$ mergeSelectivity($S_i, w_q, w_k, w_v$);
26:     end for
27:     $SF(S_j) = max_{S_i \in N_f} SF(S_i)$ ;
28:     add $S_j$ to $F_{flowId}$;
29:     remove $S_j$ from $B$;
30:     $n_v$ =getEndPoints($e^{S_j}$) $\setminus \{n_u\}$;
31:     expandFlow($S_j, n_v, flowId, w_q, w_k, w_v$);
32: end if
33: }
```

3.3 Phase 3 - Flow Cluster Refinement

The third phase of NEAT is designed to exploit opportunities to further merge some flow clusters produced from Phase 2. We describe in this section a density-based approach to group flow clusters. We modify the Hausdorff metric to measure the distance between two flow clusters and adapt DBSCAN algorithm [11] to merge flow cluster merging process. This optimization is especially effective for real time trajectory clustering where online clustering can be executed in an incremental and distributed manner. In particular, the first two phases of NEAT can be performed on each newly arrived set of trajectories. The new flow clusters are then merged with the available flow clusters to produce compact clustering results.

3.3.1 Distance Function for Flow Clusters

We modify the popular Hausdorff distance function to measure the distance between two flow clusters F_i and F_j in terms of network proximity. The distance between two flow clusters F_i and F_j can be determined by the distance between their representative routes r_{F_i} and r_{F_j}. In the first prototype of NEAT, we measure the distance between r_{F_i} and r_{F_j} by the network proximity of their ending locations. When the ends of two flow clusters are within a predefined network distance, we merge them into a larger cluster such that the resulting cluster will be able to show a group of frequent routes between two hotspot areas as illustrated in Figure 3.

Definition 11. Given two flow clusters $F_i \in \mathcal{W}$ and $F_j \in \mathcal{W}$, the distance between F_i and F_j is defined as:

$$\begin{aligned} dist_N(F_i, F_j) &= dist_N\left(r_{F_i}, r_{F_j}\right) \\ &= max\{max_{a\in\{a_1,a_2\}} min_{b\in\{b_1,b_2\}} d_N(a,b), \\ &\quad max_{b\in\{b_1,b_2\}} min_{a\in\{a_1,a_2\}} d_N(b,a)\} \end{aligned} \tag{8}$$

where $d_N(a, b)$ is the shortest path from a to b, and $\{a_1, a_2\}$, $\{b_1, b_2\}$ are the two endpoints of r_{F_i}, r_{F_j} respectively.

3.3.2 Density-based Optimization

With the modified Hausdorff distance measure for flow clusters, we need an algorithm to merge the flow clusters when their distance is within some user-defined or system-supplied default threshold. We adapt the DBSCAN algorithm to group the set of flow clusters when the density opportunity exists. The DBSCAN algorithm was originally used to cluster a set of data points and requires two parameters: a distance threshold ε between two points and a minimum number of points *minPts* in a cluster. An object is a member of a cluster if it has at least *MinPts* neighboring objects within a given radius ε. All the objects in its ε-neighborhood are also members of the same cluster. Otherwise the object is classified as noise. We make the following modifications: (1) The data unit to be clustered is a flow cluster; (2) The distance function is our modified Hausdorff distance between two flow clusters; (3) No minimum cardinality is set for the resulting cluster; (4) The density-based clustering for merging flow clusters always starts each round with the flow cluster whose representative route is the longest. It ensures that the final clustering results are always the same with the same distance threshold. This is different from DBSCAN in which data points are not processed in a deterministic order.

3.3.3 Flow-Distance Computation Optimization

As shown in Formula 8, a distance computation for each pair of flow clusters consists of four shortest path computations. Note that $d_N(a, b)$ and $d_N(b, a)$ are the same since we consider undirected graphs. Let $\mathcal{W} = \{F_1, F_2, \ldots, F_C\}$ $(C > 1)$ denote the set of flow clusters generated from Phase 2 of NEAT. For each F_i, we need to compare it with the rest $C-1$ flow clusters, one at a time, to determine whether there is a merging opportunity. When the number of flow clusters in $\mathcal{W}$ is large, the cost of computing the network proximity of a pair of flow clusters can be expensive. If we use the popular Dijkstra's network expansion algorithm to compute these shortest paths, the computation cost can be high compared to the standard Euclidean distance, especially when the representative routes of the flow clusters are long in terms of segment counts. For a graph with n nodes and m edges, traditional network expansion based algorithms (e.g. Dijkstra, Floyd-Warshall) compute shortest paths for each node pair in $O(nlogn + m)$, while the Euclidean distance computation for a node pair only takes $O(1)$.

In phase 3 of NEAT, we use the Euclidean lower bound (ELB) property to reduce the number of shortest path computations while retrieving the ε-neighborhood of a flow cluster F_i. By ELB, the Euclidean distance $d_E(l_i, l_j)$ between two road network locations l_i and l_j is always the lower bound of the network distance $d_N(l_i, l_j)$, i.e. the condition of $d_E(l_i, l_j) \leq d_N(l_i, l_j)$ is always hold. Hence, if $d_E(l_i, l_j) > \varepsilon$, we also have $d_N(l_i, l_j) > \varepsilon$. In case of $d_N(F_i, F_j)$, instead of computing four shortest paths $d_N(a_1, b_1)$, $d_N(a_1, b_2)$, $d_N(a_2, b_1)$ and $d_N(a_2, b_2)$, we compute four Euclidean between those locations first. If the minimum Euclidean distance between them exceeds ε, we can filter F_j from the search space for ε-neighborhood of F_i. Only when the minimum Euclidean distance between those points does not exceed ε, we calculate $dist_N(F_i, F_j)$ using our modified Hausdorff distance to determine whether F_j is in the ε-neighborhood of F_i or not.

3.4 Extensions

There is a number of interesting extensions of NEAT. In the first prototype of NEAT, t-fragment is the default unit for clustering. We can also integrate a user-defined fragment into our framework to let users define the granularity of fragments. For examples, a user can define the grid cells covering a road network as the building blocks in the base cluster formation phase. Then the same merging and refinement process can be applied on the resulting cell-based or any user-defined fragment clusters. In addition, in Phase 3, beside our modified Hausdorff distance, other distance functions which introduce interesting optimization results can also be included in this phase. The endpoints of the flow clusters can be considered as places of interest. Thus, we can extend our modified DBSCAN-like optimization phase to group the set of endpoints of the resulting flow clusters to discover regions of interest inherent in the given trajectory data. Furthermore, consider that each road network location is recorded with a specific timestamp, another interesting extension to NEAT is to consider the temporal information contained in the *MO* trajectories. We can apply NEAT to an extracted sub-trajectory set from the given trajectories, e.g. recorded mobility traces in rush hours or during weekends, to discover the traffic patterns during different time windows (in terms of hour, day, week, month or year). Finally, we also plan to parallelize the NEAT algorithms in future development. Specifically, in Phase 1, the same method of splitting trajectories and putting t-fragments into their associated base clusters can run for each trajectory in parallel. In Phase 2, we would divide the map into partitions and perform MapReduce-like jobs to retrieve the flow clusters. The last phase is essentially traditional clustering which can adapt many available parallelized versions of traditional clustering algorithms such as ones in the Apache Mahout project [17].

TABLE 2: Datasets used in our experiments

Datasets	**Number of points**		
	ATL	SJ	MIA
ATL/SJ/MIA500	114878	131982	276711
ATL/SJ/MIA1000	233793	255162	452224
ATL/SJ/MIA2000	468738	542598	893412
ATL/SJ/MIA3000	669924	794638	1302145
ATL/SJ/MIA5000	1277521	1296739	2262313

4 Experimental Evaluation

We perform five sets of experiments to evaluate the efficiency and effectiveness of our NEAT framework. Real road networks of different sizes are used in our experiments. In order to analyze the performance of each phase, we refer to the trajectory clustering using Phase 1 of NEAT as the base-NEAT, the trajectory clustering using the first two phases as the flow-NEAT, and the trajectory clustering using all three phases as opt-NEAT. NEAT allows users to perform trajectory clustering using any of these three versions of NEAT. Base clusters, flow clusters and final trajectory clusters are the outputs of base-NEAT, flow-NEAT and opt-NEAT respectively, and each may have its own attraction in terms of delivering interesting trajectory clustering results to location-based applications.

4.1 Experimental Setup

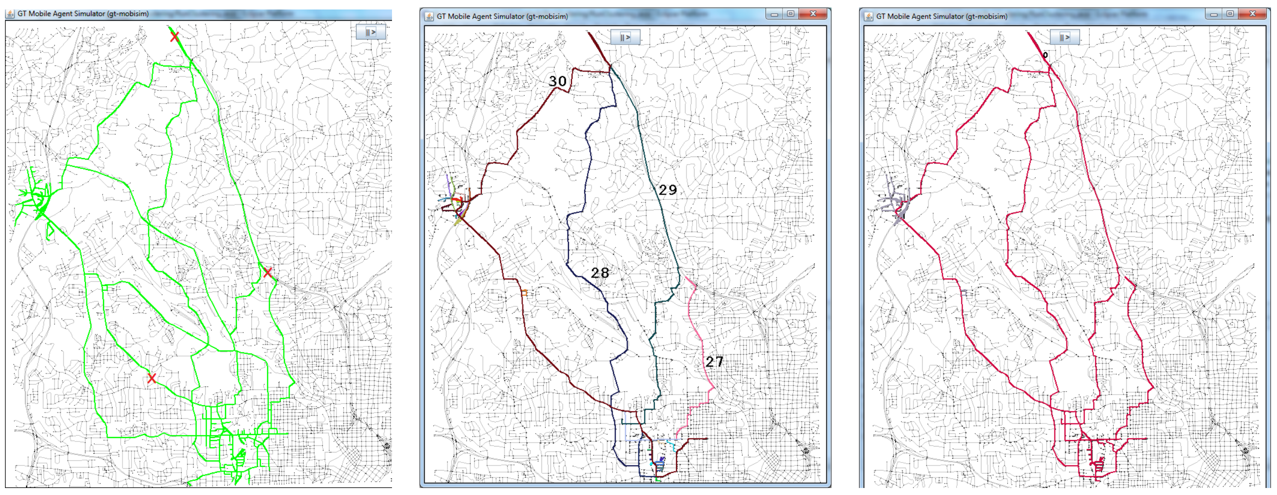
We use three real road networks in our experiments (Table 1). The road networks of North West Atlanta (ATL) and West San Jose (SJ) are obtained from [18]. The Miami-Dade (MIA) road network is obtained from [19]. We adapt the public event-based simulator GTMobiSIM [20] to generate thousands of mobility traces on those road networks for a large-scale evaluation. We use five trajectory datasets for each of the road networks ATL, SJ and MIA. Table 2 gives the information of our synthetic datasets. To create a trajectory dataset, for example SJ1000, we place 1000 mobile objects on West San Jose road network to travel under speed limit constrained on road segments, following shortest paths to a final destination chosen randomly from a predefined set of locations. We implement our algorithms using Java and visualize the results using GTMobiSIM GUI. All the experiments are conducted on the NEAT server machine with Intel Core2 Duo CPU of 2.00GHz and 1GB of main memory allocated for the Java heap size.

4.2 Visualization of NEAT clustering results

We visualize the clustering results obtained after processing trajectory datasets with the two phase NEAT approach (flow-NEAT) and with the three-phase NEAT approach (opt-NEAT). Figure 5 shows the clustering results for ATL500 dataset. Figure 5(a) plots 500 trajectories (in green color) on North West

TABLE 1: Road networks used in our experiments

Regions	Total length	# Segments	# Junctions	Avg. segment length	Junction degree
North West Atlanta, GA	1384.4km	9187	6979	150.7m	avg: 2.6, max: 6
West San Jose, CA	1821.2km	14600	10929	124.7m	avg: 2.7, max: 6
Miami-Dade, FL	26148.3km	154681	103377	169.0m	avg: 3.0, max: 9

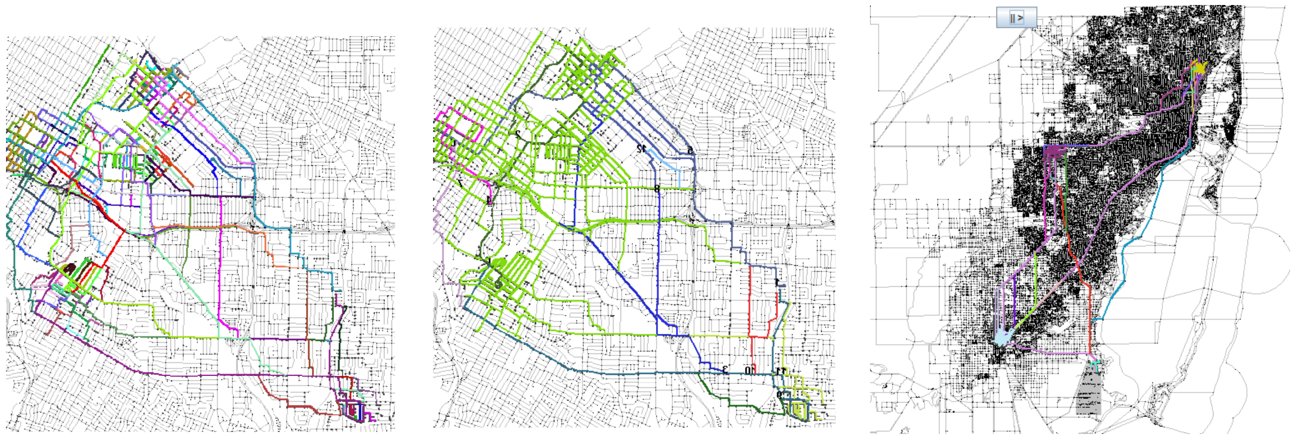

GT Mobile Agent Simulator (gt-mobisim)
GT Mobile Agent Simulator (gt-mobisim)
30
29
28
27
GT Mobile Agent Simulator (gt-mobisim)

(a) Input data: ATL500

(b) 31 flow clusters

(c) 2 clusters after optimizing phase (ε = 6500)

Fig. 5: Results for North West Atlanta road network



(a) 172 flow clusters for SJ5000

(b) 13 clusters after optimizing phase (SJ5000, $\varepsilon = 1200$)

(c) 33 clusters after optimizing phase (MIA3000, $\varepsilon = 2000$)

Fig. 6: Results for West San Jose and Miami-Dade road networks

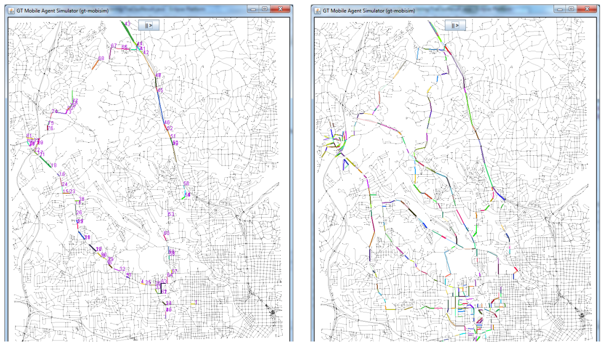
Atlanta map. After the first two phases, 31 flow clusters are discovered (Figure 5(b)). These flow clusters capture all the major traffic flows from the ATL500 dataset. Some traces that we see in the original dataset disappear in Figure 5(b) since there are too few objects moving in them. The threshold to filter those flow clusters in this experiment is $minCard$=5, which is the average number of participating trajectories in each of the flow clusters. There are two dense regions that concentrates the short flows. They are the two hotspots where we place the 500 mobile objects at the beginning of their trips. After travelling on those short flows, they start merging into the long flows to reach one of the three destinations marked with the red X-signs on the map. These 31 flow clusters are grouped into 2 clusters as shown in Figure 5(c) after performing the density-based flow cluster refinement. We start the density-based optimization with the longest representative route (the dark red polyline number 30 in Figure 5(b)). This route connects to one of the two hotspots and its endpoints are close to the endpoints of the other long flows (the polylines 29, 28 and 27). As defined in the DBSCAN algorithm, they are in a *density-connected* set. Therefore, when we perform density-based clustering (with the distance threshold ε = 6500m) we have them grouped together in one cluster (contains the red polylines in Figure 5(c)). The rest of the flows are grouped into another cluster (contains the gray polylines). The results for West San Jose road network and Miami-Dade datatsets are shown in Figure 6. For SJ5000, flow-NEAT produces 172 flows (Figure 6(a)) and opt-NEAT produces 13 clusters with $\varepsilon = 1200$m (Figure 6(b)). For MIA3000, flow-NEAT produces 300 flows and opt-NEAT produces 33 clusters with $\varepsilon = 2000$m. The visualization for MIA3000 (Figure 6(c)) is not as clear as the other two datasets because Miami-Dade road network, which is an urban area, is more dense and complicated than the ATL and SJ road networks (see Table 1 and 2).

4.3 Efficiency and Effectiveness of NEAT

In this section we evaluate the efficiency and effectiveness of NEAT by comparing it with the conventional density-based approach, represented by TraClus. Figure 7(a) shows 81 resulting clusters when applying the TraClus algorithm with ε = 10m and $MinLns$ = 30 to cluster ATL500 dataset. Each cluster has its representative trajectory plotted and numbered on the map. Another result of 460 clusters for ATL500 is shown in Figure 7(b) when applying Traclus with ε = 1m and $MinLns$ = 1. As we see, these clusters are discrete on the road network. Their representative trajectories are short in lengths. These clusters only show short routes in the road network where there is dense traffic. They do not provide information about the traffic continuity implied in the original trajectory dataset. Recall the NEAT clustering results shown in Figure 5, we can see that most of the important routes are missed when using TraClus. An interesting point to note is that our framework with base-NEAT can also provide this knowledge if we filter out those base clusters where the density is below a specific threshold. The remaining base clusters will represent the road segments where traffic is highly concentrated.

Figure 8(a) and Figure 8(b) shows the comparisons of the average and maximum lengths of the representative routes discovered using flow-NEAT and TraClus. Compared to TraClus, flow-NEAT produces clusters with longer representative routes, which are favorable for location based applications such as bus line organizing or ride sharing [9]. This results in a smaller number of clusters produced by flow-NEAT as shown in Fig 8(c). In comparison, NEAT produces more compact and meaningful results through road network aware trajectory clustering.

By utilizing the road network information, NEAT not only produces meaningful trajectory clusters but also runs very fast. Both traffic flow and traffic density can be discovered using flow-NEAT without the need to compute any distance function. We only compute shortest path distances in Phase 3 for clustering refinement and optimize this costly operation by using ELB filter to eliminate the unnecessary shortest path computation. In constrast, TraClus depends heavily on the distance measurements among every pairs of samples in the trajectory dataset. This makes TraClus overall very slow as the number of samples in each trajectory and the number of trajectories in each dataset are high. Figure 8(d) shows the efficiency comparison between the three-phase NEAT framework and



(a) 81 clusters for ATL500 (ε=10m, $MinLns$=30) (b) 460 clusters for ATL500 (ε=1m, $MinLns$=1)

Fig. 7: TraClus

TABLE 3: Number of flow clusters produced by opt-NEAT

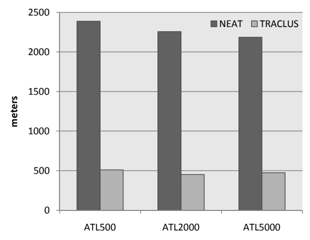
Datasets	SJ500	SJ1000	SJ2000	SJ3000	SJ5000
# flows	73	156	55	52	180

TraClus framework by varying the number of points in ATL datasets. TraClus is very time-consuming and the time complexity grows as the number of points gets larger. TraClus runs in 2573.5 seconds to cluster ATL500 (114878 points) and 334735.1 seconds to cluster ATL5000 (1277521 points). While opt-NEAT only takes 1.29 seconds to cluster ATL500 and 59.7 seconds to cluster ATL5000. Compared to TraClus, the NEAT framework is faster by more than three orders of magnitude.

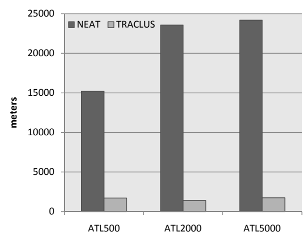
One may ask *what if TraClus is given the benefit of our map-matching preprocessing step to partition a trajectory into trajectory fragments and uses a network distance measure such as our modified Hausdorff function in its grouping phase?* To address this concern, we have run a variant of TraClus on our test datasets (the graphing result is omitted due to space limit). In this variant of TraClus, we even provide TraClus with the partitioning of trajectories into base clusters instead of t-fragments, then the grouping phase merges the base clusters using our modified Hausdorff distance. Note that the number of base clusters is usually much smaller than that of t-fragments. However, TraClus remains slow compared to NEAT due to their grouping algorithm which heavily depends on distance computations and the resulting clusters only show discrete traffic density in the road network. For instance, with the SJ2000 dataset (226151 t-fragments, 901 base clusters), this variant of TraClus took 6396.79 seconds to finish with 117 resulting clusters. While NEAT produced a more compact results of 42 flow clusters and 14 final clusters in only 11.68 seconds.

4.4 Performance of NEAT Algorithms

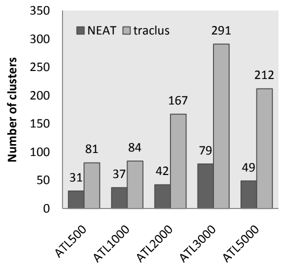
We analyze the efficiency of NEAT by analyzing the performance of different versions of NEAT during the three phases, focusing on the impact of the flow property of traffic streams in NEAT design. The scaling of base-NEAT, flow-NEAT and opt-NEAT for different MIA datasets are shown in Figure 9(a). The curves are almost linear with the growth of dataset size. In



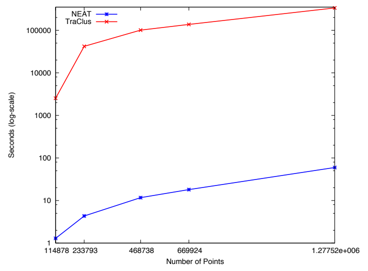
(a) Average length



(b) Maximum length

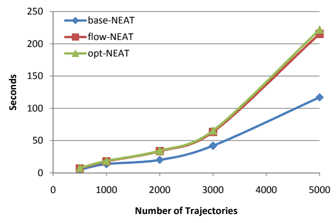


(c) Numbers of resulting clusters

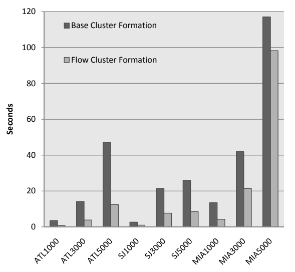


(d) Running times

Fig. 8: Comparison of flow-NEAT and TraClus (ATL datasets)

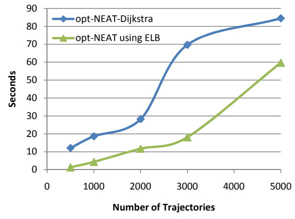


(a) MIA datasets

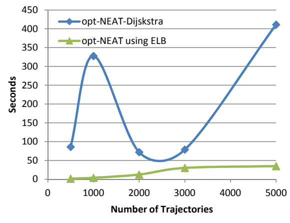


(b) Relative performance of base cluster formation and flow cluster formation

Fig. 9: Relative performance of our approaches



(a) ATL datasets



(b) SJ datasets

Fig. 10: Effectiveness of using Euclidean lower bound

all cases, the flow cluster refinement phase contributes very little to the total running time, as shown in the graphs, due to the ELB based optimization. The opt-NEAT curves nearly overlap the flow-NEAT curves.

We further investigate the relative performance of Phase 1 (base cluster formation) and Phase 2 (flow cluster formation). Phase 1 algorithm takes the road network locations as its data units because it scans the sequence of points in all the trajectories to extract t-fragments. Phase 2 algorithm takes the base cluster as its data unit. Intuitively, the number of road network locations in the trajectory dataset is much larger than the number of base clusters produced from Phase 1. Thus, it takes longer to complete Phase 1. This is confirmed by our experiments shown in Fig 9(b).

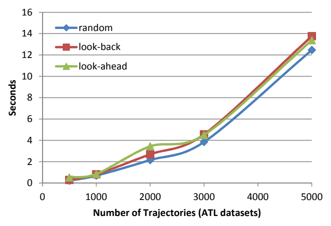
Figure 10 unveils the effectiveness of using ELB to reduce the number of shortest path computations (opt-NEAT-ELB) versus using Dijkstra's network expansion algorithm to compute all the shortest paths (opt-NEAT-Dijskstra) when performing density-based optimization in the NEAT framework. In Figure 10(a), the opt-NEAT-Dijskstra curve goes up faster as the dataset size grows. However, the curve of opt-NEAT-Dijkstra in Figure 10(b) shows that the cost at SJ1000 are much higher than at SJ2000 and SJ3000. This is due to the cost of Phase 3, which computes shortest path distances, actually depends on the number of flows produced by Phase 2 and not the data size. Table 3 shows the number of resulting flows output from the second phase where numbers of flows in SJ1000 and SJ5000 are much higher than that in other datasets for SJ road network. Using ELB significantly speeds up the performance of the density-based optimization algorithm (see some big gaps between the two curves opt-NEAT-ELB and opt-NEAT-Dijkstra).

4.5 Effects of f-neighbor Merging Decisions on Flow-based Clustering

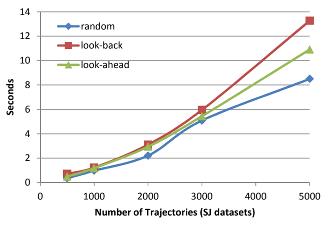
We study how using different merging schemes including "random SF_{max}" ("random" for short), "look-back" and "look-ahead" schemes, as described in Section 3.2.3, affects the results of flow-NEAT. We measure the cluster quality by computing the total netflows of the resulting flow clusters. The results are reported for ATL and SJ datasets in Figure 11 including the running time (Figures 11(a) and 11(b)) and total netflows (Figures 11(c) and 11(d)) of each scheme. It is shown that in most cases, look-head merging runs faster than look-back scheme and random merging is the fastest. Although the cost incurred by running look-ahead or look-back schemes compared to random merging is not significant, the total netflows of the resulting flow clusters in all three schemes are approximately the same. This results from the low junction degree of road networks where the average junction degree is usually less than 3. We conclude that in the context of a road network, "random SF_{max}" merging is good enough to get good clustering result and thus, NEAT uses "random SF_{max}" merging in Phase 2.

4.6 Effectiveness of Adaptive Weight Assignment on Flow-based Clustering

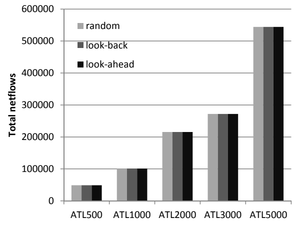
In this section, we evaluate the effectiveness of our *adaptive weight assignment* scheme (Section 3.2.4) compared to some predefined combinations of the weights (w_q, w_k, w_v) for flow factor, density factor and speed limit factor respectively. The different assignments of



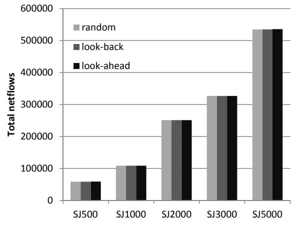
(a)



(b)



(c)



(d)

Fig. 11: Effects of using random, look-back and look-ahead f-neighbor merging schemes

TABLE 4: Total netflows

	(1/2,1/2,0)	adaptive	(1,0,0)
ATL3000	225713	271782	271986
SJ3000	275665	326723	326870
MIA3000	812505	921476	921910

TABLE 5: Total cluster density

	(1/2,1/2,0)	adaptive	(0,1,0)
ATL3000	225713	374234	374268
SJ3000	451783	475508	475536
MIA3000	1241101	1276109	1277511

(w_q, w_k, w_v) represent different merging criteria when we perform f-neighbor merging. In the previous experiments, we want to focus on the flow property of a traffic stream so we set the highest weight for the flow factor ($w_q = 1$ and $w_k = w_v = 0$), i.e. a base cluster will always be merged with its $maxFlow$-neighbor. Since the speed limit is fixed for each type of road, the speed limit factor is the least favorable among three factors. In our experiment with the adaptive weight assignment, we skip the speed limit factor by setting v_{max} value at each merging step to 0. We run flow-NEAT with this “adaptive weights” scheme and three predefined weight settings of $(1,0,0)$, $(0,1,0)$ and $(1/2,1/2,0)$. The $(1/2,1/2,0)$ weight assignment is used as a base setting where both flow and density are weighed equally. We measure the total netflows of the discovered flow clusters by running flow-NEAT using the “adaptive weights” scheme, compared to using the “netflows only” $(1,0,0)$ weight setting. The results are reported in Table 4. For ATL3000 dataset, our adaptive weights scheme achieves an increase of total netflows of 18.52% compared to the base settings $(1/2,1/2,0)$, while that of the $(1,0,0)$ setting is 20.41%. Similarly for SJ3000 and MIA3000, the percentage of total netflows increased using adaptive weights scheme is also so close to that of $(1,0,0)$ setting. With the same three datasets, Table 5 reports the total cluster density of the discovered flow clusters by running flow-NEAT using the “adaptive weights” scheme, compared to using the “density only” $(0,1,0)$ weight setting and $(1/2,1/2,0)$ setting. In all cases, the percentage of total cluster density increased using our adaptive weights scheme compared to the base setting of $(1/2,1/2,0)$ is very close to that of the $(1,0,0)$ setting (the difference is below 0.1%). Therefore, our adaptive weight assignment scheme can automatically captures highly dense and continuous traffic flow from trajectories in the road network.

5 Related Work

The clustering problem has been extensively researched in mobile ad hoc networks (MONET) and data streaming systems. In MONET, data unit to be clustered is the mobility node where the characteristics of a node are taken into account for cluster head choices in order to conserve energy and connectivity [21] [22] [23]. In data streams, k-means and density-based clustering algorithms have been extended to cluster large volume of multi-dimensional data points generated by sensor networks [24] [25] [26].

Most existing work on MO trajectory clustering [13] [27] [28] [29] [30] [31] [32] [33] has derived proximity measures for trajectories and adapted traditional k-means, hierarchical, or density-based clustering algorithms to group similar trajectories. Trajectory-OPTICS [27], which extends OPTICS algorithm [12], is a good example for grouping similar trajectories as a whole. The distance between two trajectories is the average Euclidean distance between two objects for every timestamp. Traclus [13] aims to find similar sub-trajectories rather than the whole trajectories. It partitions each trajectory into line segments using the Minimum Distance Length (MDL) principle and then performs a DBSCAN-like [11] clustering on line segments. The similarity measure is composed of three Euclidean based distance components between line segments. As a result, discovered clusters are dense regions of line segments. [31] adapts TraClus for online trajectory clustering. [32] extends TraClus for trajectory classification. However, these density-based methods cluster free space trajectories, i.e. without considering the constrained road network. Other works [28] [29] [30] consider the network constraint to derive similarity measures but they only focus on the density aspect of the given trajectories such as object density [29], common road segments [28], shortest path distance [30]. Our approach can produce partial clustering but carefully considers the constrained road network focusing on both flow and density characteristics and avoids the expensive shortest path computation in its first two phases. NEAT innovates TraClus framework in a creative manner with higher efficiency (due to reduction of network distance computation) and higher accuracy (due to incorporation of flow semantics). This paper provides a full-fledged development of the initial NEAT approach [34] with a thorough study on different merging decisions and an adaptive parameter assignment scheme capturing the most critical characteristics of a traffic stream in the core of NEAT, which allows it to discover important flow clusters in an automated manner.

6 Conclusion

We have presented NEAT - a novel road-network aware approach to *MO* trajectory clustering which clusters *MO* trajectories in a comprehensive three phase framework. We introduce the concept of f-neighborhood to identify the most critical and most interesting parts of the given *MO* trajectories *wrt.* clustering. Instead of taking points or line segments as the clustering unit as in traditional approaches, we introduce t-fragment, base cluster and flow cluster as the basic building blocks for road-network aware trajectory clustering. NEAT carefully combines the traffic locality, flow and density metrics in its three-phase trajectory clustering framework which significantly reduces the data space in each subsequent phase and ensures trajectory clustering quality. By carrying out extensive experiments, we show that NEAT discovers clusters of *MO* trajectories which represent major traffic stream in a road network and outperforms conventional density-based trajectory clustering algorithms in terms of both time complexity and accuracy.

Acknowledgment

This work is partially sponsored by grants from NSF CISE NetSE and CrossCutting program, an IBM SUR grant, an IBM faculty award and Intel ISTC.

References

[1] B. Hull et al., "CarTel: A Distributed Mobile Sensor Computing System," in *Proc. SenSys'06*.
[2] V. P. Chakka et al., "Indexing large trajectory data sets with SETI," in *Proc. CIDR'03*.
[3] S. Brakatsoulas et al., "Practical data management techniques for vehicle tracking data," in *Proc. ICDE'05*.
[4] V. Almeida et al., "Indexing the trajectories of moving objects in networks," *GeoInformatica*, vol. 9, 2005.
[5] M. Haridasan et al., "Startrack next generation: A scalable infrastructure for track-based applications," in *Proc. OSDI'10*.
[6] J. Jeong et al., "TSF: Trajectory-based statistical forwarding for infrastructure-to-vehicle data delivery in vehicular networks." in *Proc. ICDCS'10*.
[7] H. Qin et al., "An integrated network of roadside sensors and vehicles for driving safety: Concept, design and experiments." in *Proc. PerCom'10*.
[8] P. Mohan et al., "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," in *Proc. SenSys'08*.
[9] E. Kamar and E. Horvitz, "Collaboration and shared plans in the open world: studies of ridesharing," in *Proc. IJCAI'09*.
[10] The ELBA project, http://www.e-lba.com.
[11] M. Ester et al., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. SIGKDD'96*.
[12] M. Ankerst et al., "Optics: ordering points to identify the clustering structure," in *Proc. SIGMOD'99*.
[13] J. gil Lee et al., "Trajectory clustering: a partition-and-group framework," in *Proc. SIGMOD'07*.
[14] M. L. Yiu and N. Mamoulis, "Clustering objects on a spatial network," in *Proc. SIGMOD'04*.
[15] M. Weber et al., "On map matching of wireless positioning data: a selective look-ahead approach," in *Proc. GIS'10*.
[16] M. Treiber and A. Kesting, *Traffic Flow Dynamics: Data, Models and Simulation*. Springer, 2012.
[17] http://www.mahout.apache.org/.
[18] USGS data (.svg), http://edc2.usgs.gov/geodata/index.php.
[19] TIGER/Line, http://www.census.gov/geo/www/tiger.
[20] GTMobiSIM, http://code.google.com/p/gt-mobisim/.
[21] M. Chatterjee et al., "WCA: A weighted clustering algorithm for mobile ad hoc networks," *Cluster Computing*, vol. 5, 2002.
[22] Y.Wang et al., "An entropy-based weighted clustering algorithm and its optimization for ad hoc networks." in *Proc. WiMob'07*.
[23] D. Wei et al., "Clustering ad hoc networks: Schemes and classifications," *SECON'06*, vol. 3, 2006.
[24] C. Aggarwal et al., "A framework for clustering evolving data streams," in *Proc. VLDB'03*.
[25] Y. Chen et al., "Density-based clustering for real-time stream data," in *Proc. KDD'07*.
[26] P. Rodrigues et al., "Clustering distributed sensor data streams," in *Proc. ECML PKDD'08*.
[27] M. Nanni et al., "Time-focused clustering of trajectories of moving objects," *J. Intell. Inf. Syst.*, vol. 27, 2006.
[28] Jung-Im Won et al., "Trajectory clustering in road network environment." in *Proc. CIDM'09*.
[29] A. Kharrat et al., "Clustering algorithm for network constraint trajectories." in *Proc. SDH'08*.
[30] G. Roh et al., "NNCluster: An efficient clustering algorithm for road network trajectories." in *Proc. DASFAA'10*.
[31] Z. Li et al., "Incremental clustering for trajectories," in *Proc. DASFAA'10*.
[32] J. Lee et al., "Traclass: Trajectory classification using hierarchical region-based and trajectory-based clustering," in *Proc. VLDB'08*.
[33] G. Ananthanarayanan et al., "Startrack: A framework for enabling track-based applications," in *Proc. MobiSys'09*.
[34] B. Han et al., "Neat: Road network aware trajectory clustering," in *Proc. ICDCS '12*.

Binh Han received the B.Sc degree in Computer Science from Hanoi University of Science and Technology, Vietnam. She is currently a Ph.D. candidate in the School of Computer Science at Georgia Tech. She is working in the Distributed Data Intensive Systems Lab. Her research interests span mobile data management, parallel database systems and cloud computing.

Ling Liu is a full professor in the School of Computer Science at Georgia Tech. There, she directs the research programs in the Distributed Data Intensive Systems Lab (DiSL), examining various aspects of data intensive systems with a focus on performance, availability, security, privacy, and energy efficiency. She was a recipient of the best paper award at ICDCS 2003, WWW 2004, and the 2008 International Conference on Software Engineering and Data Engineering, and also won the 2005 Pat Goldberg Memorial Best Paper Award. She has served as the general chair or PC chair for numerous IEEE and ACM conferences in the data engineering, distributed computing, service computing, and cloud computing fields and is a coeditor-in-chief of the five-volume Encyclopedia of Database Systems (Springer). Currently, she is on the editorial boards of several international journals, such as Distributed and Parallel Databases (DAPD, Springer), the Journal of Parallel and Distributed Computing (JPDC), ACM Transactions on Web, IEEE Transactions on Service Computing (TSC), and Wireless Network (WINET, Springer). Her current research is primarily sponsored by the US National Science Foundation, IBM, and Intel. She is a senior member of the IEEE.

Edward Omiecinski received the PhD degree from Northwestern University in 1984. He is currently an Associate Professor at Georgia Tech in the College of Computing. He has published more than 60 papers in international journals and conferences dealing with database systems. His research has been funded by NSF, DARPA and NLM. His currently funded work deals with the discovery of knowledge in cardiac imagebases, which is a collaborative effort between Georgia Tech and Emory University researchers. He is a member of the ACM and IEEE Computer Society.