The VLDB Journal
DOI 10.1007/s00778-010-0181-y

REGULAR PAPER

Path prediction and predictive range querying in road network databases

Hoyoung Jeung · Man Lung Yiu · Xiaofang Zhou · Christian S. Jensen

Received: 3 June 2009 / Revised: 3 December 2009 / Accepted: 22 February 2010
© Springer-Verlag 2010

Abstract In automotive applications, movement-path prediction enables the delivery of predictive and relevant services to drivers, e.g., reporting traffic conditions and gas stations along the route ahead. Path prediction also enables better results of predictive range queries and reduces the location update frequency in vehicle tracking while preserving accuracy. Existing moving-object location prediction techniques in spatial-network settings largely target short-term prediction that does not extend beyond the next road junction. To go beyond short-term prediction, we formulate a network mobility model that offers a concise representation of mobility statistics extracted from massive collections of historical object trajectories. The model aims to capture the turning patterns at junctions and the travel speeds on road segments at the level of individual objects. Based on the mobility model, we present a maximum likelihood and a greedy algorithm for predicting the travel path of an object (for a time duration *h* into the future). We also present a novel and efficient server-side indexing scheme that supports predictive range queries on the mobility statistics of the objects. Empirical studies with real data suggest that our proposals are effective and efficient.

H. Jeung (✉)
School of Computer and Communication Sciences,
Ecole Polytechnique Fédérale de Lausanne (EPFL),
Lausanne, Switzerland
e-mail: hoyoung.jeung@epfl.ch

M. L. Yiu
Department of Computing, Hong Kong Polytechnic University,
Kowloon, Hong Kong
e-mail: csmlyiu@comp.polyu.edu.hk

X. Zhou
School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, Australia
e-mail: zxf@itee.uq.edu.au

C. S. Jensen
Department of Computer Science,
Aalborg University, Aalborg, Denmark
e-mail: csj@cs.aau.dk

Keywords Road network database · Path prediction · Predictive range query · Mobility statistics

1 Introduction

We are witnessing a rapid proliferation of mobile location-based services that deliver services to a user depending on the user's current location. Such services are enabled by positioning technologies such as GPS and cellular networks. Simply recognizing the user's current location is not good enough because the user needs time to plan and react. Prediction of the user's longer-term movement (e.g., 10 min in advance) with reasonable accuracy is very important to a broad range of services. This enables the services to report relevant traffic conditions and upcoming points of interest (POIs) such as gas stations. Likewise, actionable advertisements, e.g., for restaurants, should be surfaced ahead of time so that the user has plenty time to react. Path prediction also helps avoid distracting the user with irrelevant contents. In addition, it improves the quality of query results in some applications [7].

Different models for predicting the future position of a moving object have been proposed in the literature; however, they only offer accurate route predictions in the short term. A simple prediction model represents an object's future location by a *linear function* of time, based on the most recently reported location and velocity of the object. This representation is typically adopted in the context of indexing [13,25] because it is compact, easy to obtain, and reduces the amounts of updates compared with constant functions of time. However, this model does not offer accurate predictions beyond the short term. Figure 1a illustrates that linear predictions suffer from the *fork dilemma* for path prediction in a road network. The model fails to predict the object's movement at the intersection A, and its predictions (e.g., F) may well fall outside the road network.

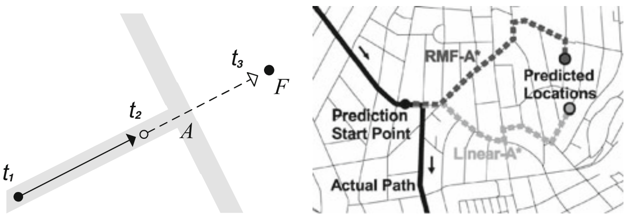
A more complex, non-linear prediction model, the *recursive motion function* (RMF) [26], achieves better predictions by finding a curve that best fits the last few reported locations of a moving object. This model, however, still suffers from the fork dilemma and fails to predict sudden direction changes (i.e., turns).

Figure 1b illustrates the application of the two prediction models to a real driver's travel 1 min into the future in a real road network. The actual route followed is shown in black. Both the linear model (gray dotted curve) and RMF (dark dotted curve) follow the object's recent motions and predict that the object will move toward the East; the routes they predict are obtained as shortest paths from the current location to corresponding predicted locations (map-matched). Clearly, both models fail to predict the sharp turn made by the object.

This paper proposes an approach that enables the accurate prediction of the route ahead of a moving object whose movement is constrained to a road network. In real life, the trajectories of pedestrians and vehicles most of the time exhibit *regular patterns* [10,16]. By taking into account an object's historical movement in a road network, we are able to accurately predict when and where the object will make a turn at an intersection.

The benefits of our approach to route prediction are threefold. First, it enables accurate computation of the *predictive path query* that predicts an object's future route up to a given time h. This allows location-based services to utilize the user's route ahead. Second, it enables accurate processing of the *predictive range query*, which identifies the objects that will fall inside a region R at h time units into the future. It is important for predicting traffic and congestion. Third, the proposed prediction model can be applied to reduce the communication cost of tracking moving objects.

The paper studies the effects of the prediction model on the accuracy of query results, quantifying the accuracy of the model as the distance and the similarity between the predicted path and the actual (future) path. It also studies the effects of the model on the frequency of location updates (i.e., the communication cost). An accurate prediction model helps reduce the frequency the location updates since the client needs not inform the server of its location as long as it follows the predicted path (which is known to the server and the client). The paper also develops an efficient server-side indexing structure for managing the mobility statistics of the objects and for efficiently processing predictive range queries.



(a) fork dilemma **(b)** predicted routes

Fig. 1 Example of path prediction on road network

The paper's contributions can be summarized as follows:

- A network mobility model that concisely represents the mobility statistics of an object, consisting of (i) turning patterns at road junctions and (ii) mined travel speed on roads.
- A maximum likelihood and a greedy algorithm for predicting the future travel path of an object moving in a road network.
- A novel and efficient server-side indexing technique for managing mobility statistics of the objects and evaluating predictive range queries.
- A comprehensive experimental evaluation that encompasses several prediction models using real datasets.

The rest of the paper is organized as follows. Section 2 covers related work, and Sect. 3 formally defines the problem addressed. Section 4 presents the network mobility model along with two path prediction algorithms for road-network constrained moving objects. Section 5 introduces an indexing technique that utilizes the mobility statistics of moving objects for answering predictive range queries. Section 6 presents the results of extensive experimental studies of path prediction and query processing using real road networks and real datasets of moving-object trajectories. Section 7 offers conclusions.

2 Related work

We first review location prediction techniques and associated predictive query processing techniques for objects moving in Euclidean space. We then cover existing techniques for tracking and predicting moving objects constrained by a road network.

2.1 Path prediction in euclidean space

Moving-object location prediction models can be classified into two types: (i) linear motion functions [13,14,22,24,25,28] and (ii) non-linear functions [1,26]. Given the location l_c and velocity v_c of an object at time t_c, the object's location at a later time h is predicted by the linear function $l(h) = l_c + v_c \cdot (h - t_c)$.

Non-linear motion functions capture an object's movements by more complex mathematical equations and generally achieve higher prediction accuracy than the simple linear model. For example, the *recursive motion function* (RMF) [26] derives the best fitting matrix-based function, defined by coefficients that reflect the object's several most recent locations (e.g., five). It is able to capture recent trends of an object's movement type (e.g., curves) for improving the prediction accuracy. RMF is good at capturing smooth and non-linear motions but incapable of predicting turns at road junctions (e.g., vehicles turning left or right).

The above motion functions fail to provide reasonable predictions for network-constrained objects because the predicted locations tend to fall outside of the network. They also all exhibit the fork dilemma as shown in Fig. 1, as well as they cannot capture sudden turns at road junctions (e.g., vehicles turning left, right, or making a u-turn).

Jeung et al. [15] propose a hybrid prediction model that utilizes both motion functions and object movement patterns for prediction. In this model, the motion functions are employed when any pattern satisfying a query condition is unavailable, thus creating the problems exhibited by the motion functions.

A *predictive range query* [18,21,22,25–28,31] asks for the objects that will appear in a given spatial region R at a future time h. Most existing predictive query processing methods employ linear motion functions for moving objects, so their results are not accurate as large h values.

Figure 2 illustrates two moving objects o_1 and o_2, and the time-parameterized rectangle (gray color) indexing them in a TPR-tree [25]. Suppose that the current time value is 1. The bounding rectangle expands with h, according to the maximum speeds of objects known at the time value 1. In the figure, the predictive range query with region R at future time $h = 3$ would retrieve the object o_1, based on its linear motion function. However, o_1 should not be a result, as its actual location at the future time $h = 3$ indeed is outside R.

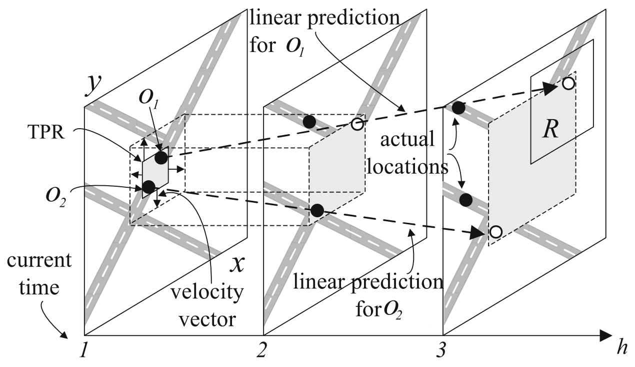


Fig. 2 An example of predictive range query processing

Moving objects on a road network (e.g., o_1, o_2) tend to turn at road junctions in the future, rendering linear predictions inaccurate in the long term. This motivates us to develop a more accurate prediction model, in order to improve the accuracy of predictive queries.

2.2 Path prediction in road networks

In order for a server to track the locations of objects in spatial networks with guaranteed accuracy at low communication cost, various techniques exist that govern the communication between the server and the moving clients [8,29]. These techniques assume that the server and a client share a prediction of the client's future movement. The client then issues an update when the deviation between its true position and the predicted one reaches the accuracy bound, upon which a new prediction is formed. The predictions utilized by these techniques usually cause the clients to issue updates every time they pass a road junction. In contrast, this paper's focus is on determining the road segments along which an object will travel beyond the next junction.

In other related work, Brilingaite et al. [5,6] predict the route of a moving object by comparing the current time and location of the object with all historical routes recorded previously for the object along with information about when they were used. This proposal appears to not scale well for a large trajectory database. Kim et al. [17] propose a path prediction method for road-network constrained moving objects as in this paper. However, they assume that the objects' destinations are known. This assumption does not hold in our setting.

A graph of cellular automata (GCA) can be used to model a road network by decomposing each road segment into a fine-grained sequence of cells. An object's movement is viewed as a sequence of transitions between cells [17]. Ding et al. [9] model a road network as a dynamic graph and the moving objects as moving graph points. However, these approaches both suffer from the fork dilemma and cannot make accurate predictions beyond a road junction.

Machine learning techniques [3,30] have also been applied to the problem of predicting object movement in road networks. The focus is here on developing effective learning models (e.g., utilizing neural networks and Markov models); little attention is given to providing mechanisms for estimating the path of an object or its location at a specific future time.

In the context of a wireless cellular communication networks, methods [19,23] are available that aim to predict the next cell to be visited by a moving object by using signal strengths from the wireless base stations. However, cells may span many square miles and usually contain many different road segments, rendering the predictions too coarse grained for our purposes.

The work most relevant to this paper's is the PLM prediction model [16,20]. In a pre-processing phase, historical trajectories of an object are used to formulate a probabilistic model for capturing the object's turning behavior at road junctions.

Figure 3 illustrates an object's actual path and the path predicted by the PLM model for a predictive time of $h = 2$ min, starting from the location q.

Next, the system defines a circle based on q, the value h, and the speed limit of the road segment of q. Intersecting the circle with the road segments, we obtain a set of *exit points* in the network (i.e., the small circles in the figure). By performing a depth-first graph search starting at q, we obtain a path from q to each exit point. The probability of each path is computed by multiplying the turning probabilities at all road junctions along a path. The path with the highest probability is the predicted path.

This approach has two drawbacks. First, it may return an inaccurate predicted path. A depth-first search is performed without considering the turning probabilities at road junctions. As a result, the predicted path is generally not an intuitive route for a driver (e.g., as suggested in Fig. 3). Although a predicted path has the highest probability among all paths to an exit point, it is not guaranteed to be the path that maximizes the probability to an exit point. Furthermore, the search range of PLM is based on the Euclidean distance obtained by multiplying the time h by the maximum speed of the object's current road segment. However, depending on the road segments traveled, the speed of the object may deviate significantly from this maximum speed. The object may not actually be able to reach any of the exit points by time h, or it might even move beyond the large circle.

Second, the method can incur high computational cost during the prediction process since it needs to compute all exit points by intersecting the circular range with the set of road segments. For typical values of h, the number of exit points is high, implying a high computational cost. For instance, $h = 2$ yields 96 exit points in Fig. 3. As a result, considering all possible paths to the exit points is a costly proposition.

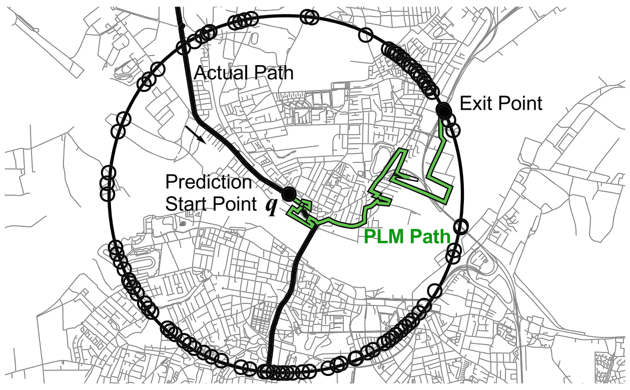


Fig. 3 An example of the PLM method

3 Problem setting

In this section, we formally define a road network as well as the distance measures and queries considered; we end by introducing the system architecture. Table 1 offers the notation that is introduced in this section and that will be used throughout the paper.

3.1 Road network model and distance notions

A *road network* is an undirected, weighted graph $G = (V, E, W, A)$, where V is a set of vertices, E is a set of edges (among vertices in V), W is a function that maps each edge to a weight (i.e., length of road segment), and A is a function that maps each edge to a set of descriptive attribute values.

Table 2 shows an example of the information maintained for edges. A may capture descriptive attributes such as speed limits for the road segments.

In the network model, a tuple $p = (e, d)$ represents a location in the network that falls on the edge $e = (v_i, v_j)$ at the distance d from v_i. Let $W(e)$ be the weight (length)

Table 1 Summary of notations

Symbol	Meaning
$e = (v_i, v_j)$	Edge connecting two vertices v_i and v_j
d	Distance from a position on $e = (v_i, v_j)$ to v_i
t	Time value
$p = (e, d, t)$	Time stamped network position
h	Prediction time length
$D(p, q)$	Network distance between network positions p, q
$D_L(p, v)$	Distance from a position p to a vertex v
$D_P(c_i, c_j)$	Prediction distance from grid cell c_i to c_j
$S(e), W(e)$	Estimated travel speed and weight of edge e
$\mathcal{M}(v), \mathcal{M}_R(v)$	Mobility and reverse mobility statistics for v

Table 2 Description examples of edges

Edge e_i	End vertices (v_i, v_j)	Weight (length) $W(e_i)$	Speed limit $A(e_i)$
e_1	(v_1, v_2)	70.2	60
e_2	(v_2, v_3)	70.8	80
e_3	(v_1, v_4)	100.1	40
e_4	(v_2, v_4)	20.6	60
e_5	(v_4, v_5)	50.3	60

of edge e. Then the location of vertices v_i and v_j can be represented as $(e, 0)$ and $(e, W(e))$, respectively.

We proceed to define the notions of distance over a road network. The *network distance* $D(v, v')$ between two vertices v and v' is defined as the length of the shortest path between v and v', i.e., as the sum of the weights associated with the edges that make up the path.

Given a location $p = (e, d)$ on $e = (v_i, v_j)$, its *distance* from v_i is defined as $D_L(p, v_i) = d$. Similarly, its distance from v_j is defined as $D_L(p, v_j) = W(e) - d$.

Given network positions p and p' on the edges (v_i, v_j) and $(v_{i'}, v_{j'})$, respectively, we define their *network distance* as

$$D(p, p') = \min_{a \in \{i,j\}, b \in \{i',j'\}} (D_L(p, v_a) + D(v_a, v_b) + D_L(v_b, p'))$$

The *network trajectory* of an object o is a sequence of timestamped locations $o_t = (e, d, t)$ ordered ascendingly on t, where t is the timestamp. Table 3 lists four examples of network trajectories.

3.2 Predictive queries

We proceed to define two types of predictive queries along with error metrics for measuring the quality of a prediction model.

Definition 1 Given the location $q_c = (e, d, t_c)$ of an object (at the current time t_c), its next vertex to be visited, and a prediction time length h, the **predictive path query** returns a path of the object within the time interval $[t_c, t_c + h]$.

Since our network model is undirected, we obtain the direction that the query object is heading to from the next vertex to be visited.

We consider two error measures for the predictive path query, namely *distance error* and *path similarity*. Let q_{act} be the actual location of the object at time $t_c + h$ (which will become known only in the future), and let q_{pred} be the location of the object in the predicted path at time $t_c + h$ (returned by a prediction model). We define the distance error of the prediction model (for the query) as $D(q_{act}, q_{pred})$, i.e., the network distance between q_{act} and q_{pred}.

Table 3 Examples of network trajectories

Trajectory ID	Sequence of network locations
1	$(e_1, 10.4, t_1), (e_1, 30.5, t_2), (e_2, 20.1, t_3)$
2	$(e_1, 20.2, t_2), (e_1, 35.9, t_3)$
3	$(e_3, 70.8, t_1), (e_5, 15.3, t_2), (e_5, 50.2, t_3)$
4	$(e_1, 40.6, t_1), (e_1, 30.1, t_2), (e_2, 20.7, t_3)$

In some cases, the distance error is small while the predicted path is very different from the actual path. To capture this, we also take into account the path similarity. Let E_{act} be the set of edges that the actual path from q_c to q_{act} contains, and let E_{pred} be the set of edges that the predicted path from q_c to q_{pred} contains.

We define the *precision* and *recall* as follows:

$$Pre(E_{act}, E_{pred}) = \frac{|E_{act} \cap E_{pred}|}{|E_{pred}|}$$

$$Rec(E_{act}, E_{pred}) = \frac{|E_{act} \cap E_{pred}|}{|E_{act}|}$$

By combining the two, we measure the *path similarity* as the *F1-score* [4] of the prediction model:

$$F1(E_{act}, E_{pred}) = \frac{2 \cdot Pre(E_{act}, E_{pred}) \cdot Rec(E_{act}, E_{pred})}{Pre(E_{act}, E_{pred}) + Rec(E_{act}, E_{pred})}$$

Note that merely counting the number of common edges between an actual path and a predicted path is not effective in capturing cases where a predicted path has a predicted path has a very large number of edges that contain most edges of the actual path, while the actual path is short, and the two paths are dissimilar.

Next, we define the second type of predictive query:

Definition 2 Given a set of objects O (each with its location as of the current time t_c and its next vertex to be visited), a spatial region R, and a prediction time length h, the **predictive range query** returns the set of objects in region R at time $t_c + h$.

We also measure the quality of a predictive range query result with the F1-score. Let S_{act} be the actual set of objects in R at time $t_c + h$ (know only in the future), and let S_{pred} be the predicted set of objects in R at time $t_c + h$ (returned by the prediction model). The F1-score for the predictive range query then becomes:

$$F1(S_{act}, S_{pred}) = \frac{2 \cdot Pre(S_{act}, S_{pred}) \cdot Rec(S_{act}, S_{pred})}{Pre(S_{act}, S_{pred}) + Rec(S_{act}, S_{pred})}$$

Our objective is to design prediction models for objects moving in road networks that minimize the distance error and maximize the F1-score of query results. In addition, the prediction model should be conducive to efficient computation.

3.3 Architecture and supported functionality

Our proposed solution assumes a standard client–server architecture, encompassing a *location-based service provider* (LBS). The LBS has two functionalities: (i) tracking the locations of its mobile clients (objects) and (ii) processing queries (e.g., predictive path and range queries).

Table 4 Supported operations and their quality measures

Operation	Quality measure
Predictive path query	Distance error, path similarity (F1-score)
Predictive range query	F1-score
Location tracking	Location update frequency

Given a distance error threshold ϵ (a system parameter), the LBS and the clients implement a *location tracking* method [8] such that the location of each object maintained at the server is guaranteed to be within distance ϵ of its actual current location. For this purpose, both the client and the server share the same prediction model. Each client is equipped with a GPS device and reports its current location q to the LBS when the distance between its actual current location and its predicted location (by the prediction model) reaches ϵ. Observe that the prediction model plays a crucial role in location tracking. We measure *the frequency of location updates* in order to capture the quality of the prediction model. An accurate path prediction model will substantially reduce the frequency of location updates from the client.

Table 4 summarizes the operations supported by our proposed solution, as well as their corresponding quality measures.

4 Client-side prediction

We first present a network mobility model that aims to capture driving patterns. Based on the network mobility model, we then present a *Maximum Likelihood* and a *Greedy* algorithm for predicting the travel path of an object, without knowing the object's destination.

4.1 Network mobility model

To predict the path of a moving object, we introduce a practical estimation model for predicting two key factors, the *directions* in which the object will turn at junctions and the *travel speeds* on the road segments.

4.1.1 Turning patterns at road junctions

Vehicles exhibit turning patterns at road junctions. For example, a regular commuter driving to work usually turns in particular directions at certain junctions. After loading goods from factories, delivery trucks may make specific turns at junctions in order to enter highways. These patterns serve as important background information and help us address the fork dilemma.

As an example, Fig. 4 shows the trajectories of an object over four days. On three of those days, the object made the

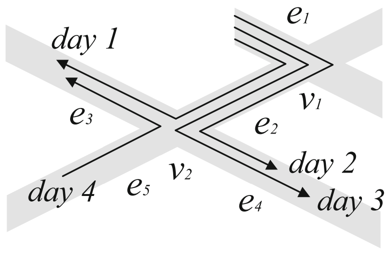


Fig. 4 Typical turning patterns at road junctions

same turn at the junction v_1. In case the query object is now traveling on segment e_1 toward v_1, we predict that it will again make the same turn. Likewise, when the object is moving on e_2 toward v_2, it has the highest probability of turning left ($\frac{2}{3}$), the second highest of turning right ($\frac{1}{3}$), and the lowest of going straight (0). These values, we term individual *mobility statistics*.

Motivated by the above observations, we propose a *network mobility model* that models the turning patterns of objects at junctions by means of probabilities. Let $deg(v)$ be the degree of a vertex v. The predictions of a turn at any degree-1 vertex (i.e., a dead-end) and any degree-2 vertex (i.e., no turning options) are straightforward. Thus, we consider only vertices with a degree above 2 (i.e., junctions) in the sequel.

To extract statistics from the historical trajectories, we first define the support count of a pair of adjacent edges:

Definition 3 Let $\mathcal{D}_u$ be a set of (historical) network trajectories (of the user u). Given two adjacent edges $e_i \succ e_j$, we define the **support** $\tau(\mathcal{D}_u, e_i \succ e_j)$ as the number of trajectories in $\mathcal{D}_u$ that contain $e_i \succ e_j$.

Given two edges e_i and e_j incident to the vertex v, we define the *transition probability* from e_i to e_j as:

$$\mathcal{M}(v)[e_i, e_j] = \frac{\tau(\mathcal{D}_u, e_i \succ e_j)}{\sum_{e_k} \tau(\mathcal{D}_u, e_i \succ e_k)} \tag{1}$$

Note that our transition probabilities on v are built purely based on the histories of the objects that passed through v, while some previous work (e.g., the PLM method [16]) considers the degree of the vertex $deg(v)$ when defining transition probabilities. One advantage of our approach over the previous approaches is that only non-zero entries in $\mathcal{M}(v)$ need to be kept, whereas the previous approaches need to keep $deg(v) \cdot (deg(v) - 1)$ entries, which is the worst-case size in our definition.

In Eq. 1, a small denominator value (e.g., 1) could lead to a high transition probability $\mathcal{M}(v)[e_i, e_j]$, which may mislead the prediction process. As in association rule mining [2], we therefore specify two system parameters: (i) support threshold s_{thr} and (ii) confidence threshold c_{thr}. A transition probability $\mathcal{M}(v)[e_i, e_j]$ is said to be *valid* if it satisfies both conditions: (i) $\tau(\mathcal{D}_u, e_i \succ e_j) \geq s_{thr}$ and (ii)

$\mathcal{M}(v)[e_i, e_j] \geq c_{thr}$. Otherwise, the value $\mathcal{M}(v)[e_i, e_j]$ is regarded as zero and is not stored in the system. For example, in Fig. 4, the user's individual mobility statistics at v_2 are computed as: $\mathcal{M}(v_2)[e_2, e_3] = \frac{1}{3}$, $\mathcal{M}(v_2)[e_2, e_4] = \frac{2}{3}$, $\mathcal{M}(v_2)[e_5, e_3] = \frac{1}{1}$.

We expect that $\mathcal{M}$ captures the specific behavior of a particular user and tends to provide accurate predictions. Nevertheless, some entries of $\mathcal{M}(v)$ may still be empty. For example, in Fig. 4, all entries of the form $\mathcal{M}(v_2)[e_4, *]$ are empty. In case the query object reaches v_2 from road segment e_4, we cannot determine which road segment (e.g., e_2, e_3, or e_5) the object will travel next. To complement such "empty" entries, we define the notion of *reverse mobility statistics* $\mathcal{M}_R(v)[e_i, e_j]$:

$$\mathcal{M}_R(v)[e_i, e_j] = \frac{\tau(\mathcal{D}_u, e_j \succ e_i)}{\sum_{e_k} \tau(\mathcal{D}_u, e_k \succ e_i)} \tag{2}$$

In the example of Fig. 4, we obtain $\mathcal{M}_R(v_2)[e_4, e_2] = \frac{2}{2}$.

These reverse mobility statistics reflect the object's movement tendency to a certain extent. Suppose that a driver leaves home and visits a new place where the driver has never been. The driver then is likely to go back home with the same route, because the driver may not want to take roads that she does not know. In this example, the reverse mobility statistics serve as good supports for the prediction of routes, rather than selecting random directions on road junctions. In later sections, we will utilize these reverse mobility statistics (Eq. 2) for best-effort prediction, in case mobility statistics (Eq. 1) are unavailable.

4.1.2 Travel speed estimation

Another important aspect of the prediction problem is to predict the future speed of an object. The challenge is that an object seldom maintains a constant travel speed; its speed is influenced by various factors, including the road network and the traffic volume.

Considering an edge e, a natural idea is to mine the average historical travel speed $S^*(e)$ from the trajectory dataset $\mathcal{D}_u$ (of the user) by examining the speeds of the sub-trajectories that correspond to edge e. As a complication, $S^*(e)$ is undefined if no such trajectories exist. We may then derive the travel speed $S(e)$ on edge e by utilizing the edge attributes introduced in Sect. 3. The specific design of formulas for computing these speeds is beyond the scope of this paper; the transportation research literature offers proposals. Here, we give a heuristic example of modeling $S(e)$ using its speed limit $A(e)$ and a coefficient α (a system parameter). Thus, the travel speed $S(e)$ on a road segment e is defined as:

$$S(e) = \begin{cases} S^*(e) & \text{if } S^*(e) \text{ exists} \\ A(e) \cdot \alpha & \text{otherwise} \end{cases}$$

In the experimental study, we compare the effects of different travel speed function on the prediction accuracy: (i) the mined speed $S^*(e)$, (ii) the maximum speed (i.e., $\alpha = 1$), and (iii) the half maximum speed (i.e., $\alpha = \frac{1}{2}$).

4.2 Maximum likelihood path prediction

In this section, we develop a prediction algorithm that determines the predicted path purely based on the probability values stored in the $\mathcal{M}(v)[e_i, e_j]$. In case of an empty $\mathcal{M}(v)[e_i, e_j]$ entry, we set $\mathcal{M}(v)[e_i, e_j]$ to the default value $\frac{1}{deg(v)-1}$ as previous studies do [16,20]. This means that each outgoing edge e_j has the same probability of being traveled next. Note that we do not use the reverse mobility statistics $\mathcal{M}_R(v)[e_i, e_j]$ here.

4.2.1 Foundation

Let $\mathcal{P} : v_1 \succ v_2 \succ \cdots \succ v_k$ be a path (i.e., a consecutive sequence of vertices), where e_i is the edge between vertices v_i and v_{i+1}. We define the *travel probability* of the path $\mathcal{P}$ as follows:

$$Pr(\mathcal{P}) = \prod_{i=1}^{k-1} \mathcal{M}(v_i)[e_i, e_{i+1}] \tag{3}$$

We now define the concept of *maximum travel probability path*:

Definition 4 The path $\mathcal{P} : v_1 \succ v_2 \succ \cdots \succ v_k$ is said to be the *maximum travel probability path* between the vertices v_s and v_e, if $v_s = v_1$, $v_k = v_e$, and the value $Pr(\mathcal{P})$ is maximized among all possible paths between v_s and v_e.

The following lemma states an interesting property about maximum travel probability path $\mathcal{P}$, namely that every subpath of $\mathcal{P}$ must also be a maximum travel probability path in its own right.

Lemma 1 Maximum travel probability path property *Given two vertices $v_s (= v_1)$ and $v_e (= v_k)$, let $\mathcal{P}$: $v_1 \succ v_2 \succ \cdots \succ v_k$ be the maximum travel probability path between them. It follows that for any vertices v_i and $v_j (i < j)$, the path $\mathcal{P}' : v_i \succ v_{i+i} \succ \cdots \succ v_j$ is the maximum travel probability path between v_i and v_j.*

Proof For the sake of contradiction, assume that there exists $i, j (i < j)$ and a path $\mathcal{P}'' : v_i \succ \cdots \succ v_j$ such that $Pr(\mathcal{P}'') > Pr(\mathcal{P}')$. By using the path $\mathcal{P}''$, we construct a new path $\mathcal{P}^*$ such that $\mathcal{P}^*$: $v_1 \succ \cdots \succ v_{i-1} \succ \mathcal{P}'' \succ v_{j+1} \succ \cdots \succ v_k$. Thus, we have $Pr(\mathcal{P}^*) = Pr(\mathcal{P})/Pr(\mathcal{P}') \cdot Pr(\mathcal{P}'') > Pr(\mathcal{P})$. □

4.2.2 Prediction algorithm

We proceed to present the *Maximum Likelihood* algorithm for path prediction. According to Lemma 1, once a path is known to be a maximum travel probability path, any fragment of the path is also a maximum travel probability path. This implies that we can find the maximum travel probability path by conquering the fragments (i.e., sub-paths) gradually, using a bottom up approach.

This idea can be realized by employing the *best-first* graph search. Specifically, we define the evaluation function for the best-first search as the total probability of a given sub-path. We then always choose the sub-path having the highest score from the function, at every step forward in the graph search. This approach is similar to how shortest path algorithms work and thus has the same time complexity (see Table 5) as general shortest path computation.

Despite having the highest value, the maximum travel probability may become small when the path becomes long. However, the algorithm uses only the probabilities as relative values for selecting the best path for prediction—the absolute values are not given significance. Therefore, the travel probabilities carry different semantics from the probability of "correctness" that a user actually follows the maximum travel probability path. This correctness probability is reflected by path similarity defined in Sect. 3. We show that the path similarity for this algorithm is reasonably high in our experiments (Sect. 6), regardless of the absolute maximum travel probability values involved.

Algorithm 1 shows the pseudo-code of the Maximum Likelihood algorithm. The user specifies a current location l and a prediction time h. In addition, the algorithm takes the network graph G and the mobility statistics $\mathcal{M}$ as inputs. The result of the algorithm is a predicted path $\mathcal{P}$, i.e., a sequence of timestamped locations on the network. In Lines 1–3, we set the current edge e_{cur} to the edge on which l resides, and we set the current vertex v_{cur} to be the vertex ahead. The expected travel time h' on the current edge is also computed.

In Lines 4–7, we also initialize V that describes the visited vertices, which is registering the last vertex in order to avoid u-turns. We then compose an initial entry for the priority queue that includes *edges*, a sequence of edges visited, v_{cur}, the vertex being approached, p, the total probability of the current sub-path (*edges*), and h', the expected travel time so far. Note that the priority of the entries in the queue Q is imposed by the total probability p associated with *edges*.

In Line 9, a queue entry that has the highest probability is picked first, which is the key difference from the PLM method. In Lines 13–16, the algorithm then determines whether the entry q's travel time is no smaller than the given prediction time. If so, the predicted path is composed and returned.

Algorithm 1 Maximum likelihood $(G, l, h, \mathcal{M})$

```
Input: graph $G$, current location $l = (e, d, t)$, prediction length $h$,
    network mobility model $\mathcal{M}$
Output: a predicted path $\mathcal{P}$ (i.e. timestamped network locations)
 1: $e_{cur} \leftarrow l.e$
 2: $v_{cur} \leftarrow$ end vertex of $e_{cur}$, being approached by the object
 3: expected travel time $h' \leftarrow \frac{D_L(l, v_{cur})}{S(e_{cur})}$
 4: visited vertices $V \leftarrow$ register the other end vertex of $e_{cur}$ from $v_{cur}$
 5: predicted edge sequence $edges \leftarrow$ append $e_{cur}$
 6: path probability $p \leftarrow 1.0$
 7: priority queue $Q \leftarrow$ push an entry $\{edges, v_{cur}, p, h'\}$
 8: while $Q \neq \emptyset$ do
 9:   pop an entry $q$ (with the highest path probability) from $Q$
10:   $V \leftarrow$ register $q.v_{cur}$
11:   $e_{cur} \leftarrow$ the last edge of $q.edges$
12:   $v_{cur} \leftarrow q.v_{cur}$
13:   if $q.h' \geq h$ then
14:     $\mathcal{P} \leftarrow$ convert $q.edges$ to network positions, except $e_{cur}$
15:     $\mathcal{P} \leftarrow$ append $(e_{cur}, W(e_{cur}) - S(e_{cur}) \cdot (q.h' - h), l.t + h)$
16:     return $\mathcal{P}$
17:   edge set $E_{next} \leftarrow$ incident edges to $v_{cur}$, except $e_{cur}$
18:   for each $e_{next} \in E_{next}$ do
19:     $v_{next} \leftarrow$ the other end vertex of $e_{next}$ from $v_{cur}$
20:     if $v_{next}$ is not registered in $V$, and $deg(v_{next}) \neq 1$ then
21:       a new priority queue entry $q' \leftarrow$ copy $q$
22:       if $deg(v_{cur}) > 2$ then
23:         $q'.p \leftarrow q'.p \times \mathcal{M}(v_{cur})[e_{cur}, e_{next}]$
24:       $q'.edges \leftarrow$ append $e_{next}$
25:       $q'.v_{cur} \leftarrow v_{next}$
26:       $q'.h' \leftarrow q'.h' + \frac{W(e_{next})}{S(e_{next})}$
27:       insert $q'$ into $Q$
```

In Lines 18–27, the queue entry is updated by considering neighbor edges of the current one, unless the end vertex of each neighbor has not yet been visited. Due to the "if" statement in Line 20, the algorithm never enters a cycle. In addition, the algorithm never follows a path that goes to a dead-end (i.e., $deg(v_{next}) = 1$ in Line 20). Such a path will have the same probability no matter how many iterations occur, while the probabilities of other paths will keep decreasing due to multiplication. This would otherwise prevent selecting a proper (long) path, especially for long-term predictions. For example, suppose that there are two paths $\mathcal{P}_1 : v_1 \succ v_2$ and $\mathcal{P}_2 : v_1 \succ v_3 \succ v_4 \succ \cdots \succ v_k$, where v_2 is a dead-end and $\mathcal{P}_2$ is the actual path that an object follows. Since the number of vertices involved in $\mathcal{P}_2$ is greater than for $\mathcal{P}_1$, the travel probability for $\mathcal{P}_2$ is likely to be lower than that for $\mathcal{P}_2$ due to the high number of probability multiplication caused by the vertices in $\mathcal{P}_2$. Thus, if the algorithm was to accept dead-ends, this would decrease the probability that the actual path $\mathcal{P}_2$ would be selected as the predicted path.

Next, a new entry q' for Q is composed. If a fork dilemma occurs on the current vertex (i.e., $deg(v_{cur}) > 2$), its path probability is updated (Line 23). The algorithm then updates each element of the new entry, and the entry is pushed to the priority queue. The entry may be processed in the next iteration if it has the highest probability among all existing entries in Q.

4.2.3 Example

Figure 5 exemplifies the Maximum Likelihood algorithm with an object's current location at l, the vertex that this object is approaching being v_1, and prediction length $h = 5$.

First, we estimate the expected travel time (h') from the location l to the vertex v_1. If h' is smaller than the given prediction time h, we *transit* to the next edge having the highest probability. Since $\mathcal{M}(v_1)[e_1, e_2] > \mathcal{M}(v_1)[e_1, e_5]$, we select e_2 and recompute h' for that edge after including the travel time of the visited edges. Until v_2, the value for h' is still smaller than h, but $\mathcal{M}(v_2)$ is not available. Hence, the probability for each next possible edge is assigned equally (i.e., 0.5).

So far, the probabilities for both sub-paths $e_1 \succ e_2 \succ e_3$ and $e_1 \succ e_2 \succ e_4$ are $0.55 \cdot 0.5 = 0.275$, and that for sub-path $e_1 \succ e_5$ is 0.45. Therefore, the Maximum Likelihood method examines the sub-path $e_1 \succ e_5$ instead of further investigating the previous sub-paths, since $e_1 \succ e_5$ has the highest path probability (i.e., 0.45) among all passible sub-paths that have been examined. At v_5, no fork–dilemma occurs; thus, the process passes on to e_6, and the probability of $e_1 \succ e_5 \succ e_6$ is the same. Next, it considers the probabilities for sub-paths $e_1 \succ e_5 \succ e_6 \succ e_7$ and $e_1 \succ e_5 \succ e_6 \succ e_8$, which are $0.45 \cdot 1.0 \cdot 0.4 = 0.18$ and $0.45 \cdot 1.0 \cdot 0.6 = 0.27$, respectively.

These probabilities are smaller than the probability of $e_1 \succ e_2 \succ e_3(e_4)$; hence we are now back to the consideration of the sub-paths $e_1 \succ e_2 \succ e_3$ and $e_1 \succ e_2 \succ e_4$. While the h' value for $e_1 \succ e_2 \succ e_3$ is still smaller than h, that for $e_1 \succ e_2 \succ e_4$ is greater than h. Finally, we compute the forecasted location on edge e_4 and return $e_1 \succ e_2 \succ e_4$ as the predicted path.

$$\mathcal{M}(v_1) = \begin{bmatrix} e_1, e_2 : 0.55 \\ e_1, e_5 : 0.45 \end{bmatrix} \qquad \mathcal{M}(v_6) = \begin{bmatrix} e_6, e_7 : 0.4 \\ e_6, e_8 : 0.6 \end{bmatrix}$$

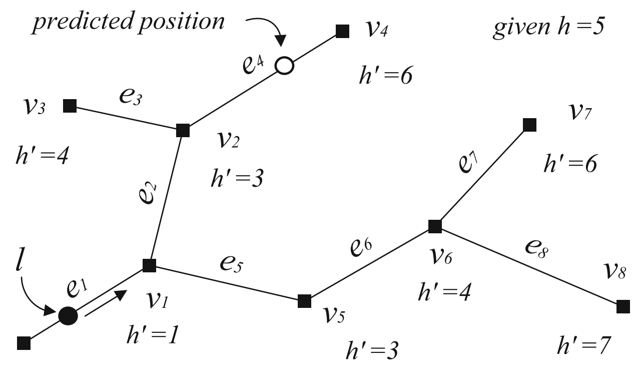


Fig. 5 An example of maximum likelihood path prediction

4.2.4 Comparison with PLM

It is instructive to compare the features of Maximum Likelihood and PLM. The main advantage of Maximum Likelihood over PLM is that Maximum Likelihood guarantees that the returned path maximizes the travel probability over all possible paths. In contrast, PLM is restricted to return a path that belongs to the set of paths traversed by the depth-first search. Therefore, the selected path is not necessarily the one with the maximum travel probability across all possible paths.

In addition, Maximum Likelihood considers the travel speed and network distance of each road segment, whereas PLM computes the exit points based on the Euclidean distance and the maximum travel speed of the segment of the starting location, which may yield a distance that does not approximate well how far the object can actually travel.

The Maximum Likelihood algorithm avoids unnecessary computations, as it terminates as soon as it determines that no other path with higher probability (at prediction time h) can be found. PLM attempts to find all possible paths to every exit point, without considering the probability. Notice that the number of exit points becomes very large when h is large.

4.3 Greedy path prediction

While the Maximum Likelihood algorithm returns a path guaranteed to have the highest probability, its performance may not scale well for long-term prediction. When the prediction length h is long, the search space of the graph for Maximum Likelihood becomes large, which yields many of sub-paths to be compared. This may limit the utility of Maximum Likelihood for applications that need high efficiency.

Motivated by this, we develop a path prediction algorithm named *Greedy* that achieves high efficiency and low memory use, while facilitating near-maximum probability. The core idea of Greedy is also based on Lemma 1. Assuming that all vertices have mobility statistics, we can find the maximum probability path by progressively selecting a next edge having the highest transition probability of each vertex, instead of considering all other possible sub-paths. In practice, however, the assumption does not always hold, and a problem arises when Greedy faces at a vertex whose mobility statistics are unavailable. While Maximum Likelihood can examine the next "future" vertices and use them for making a decision on the current vertex, Greedy lacks this ability.

We consider two alternative means of addressing this problem. First, we take into account reverse mobility statistics for the selection of the next edge. Although the reverse probabilities carry different semantics, they still reflect the driving patterns of individuals. Second, in case the reverse mobility statistics are also unavailable, we select as the next edge having the smallest deviation angle from the object's current travel direction, which is derived from the object's initial position and the current position. People generally drive to certain destinations, and they are likely to prefer to move in the general directions of the destinations. While our work assumes that the destination of an object is not given, the current travel direction reflects the ultimate direction to the (unknown) final destination. Hence, the current direction may serve as a better means of making a decision than selecting the next edge at random.

Based on the above ideas, we presents the pseudo-code of Greedy in Algorithm 2. The query inputs, the return value, and the initialization of variables in Lines 1–3 are the same as those in Algorithm 1.

In the loop in Lines 4–20, we traverse the graph iteratively until the expected travel time h' exceeds the given prediction length h. In case of a dead-end road (i.e., $deg(v_{cur}) = 1$), the algorithm terminates and returns a corresponding location on the edge as the final prediction position (Lines 6–8). This differs from how Algorithm 1 works, since Maximum Likelihood does not capture any position on dead-end roads as a predicted location. When the degree of v_{cur} is 2, there is only one adjacent edge for transition, and thus the algorithm transits to that edge (Lines 9–10).

In Lines 11–17, the algorithm handles the fork dilemma, as it predicts where the object will turn at the junction v_{cur}. It first attempts to utilize the mobility statistics $\mathcal{M}(v_{cur})$; in case these are unavailable, it determines whether any reverse mobility pattern ($\mathcal{M}_R(v_{cur})$) exists. If this is also unavailable, it selects the edge having the smallest deviation angle from the object's current travel direction among all possible edges.

From Lines 18 to 20, the current edge e_{cur}, vertex v_{cur}, and travel time are updated for the next iteration. After completing the loop, the final location is computed according to the travel time, and the predicted path is returned.

Algorithm 2 Greedy $(G, l, h, \mathcal{M})$

```
Input: graph $G$, current location $l = (e, d, t)$, prediction length $h$,
    network mobility model $\mathcal{M}$
Output: a predicted path $\mathcal{P}$ (i.e. timestamped network locations)
1: $e_{cur} \leftarrow l.e$
2: $v_{cur} \leftarrow$ end vertex of $e_{cur}$, being approached by the object
3: expected travel time $h' \leftarrow \frac{D_L(l, v_{cur})}{S(e_{cur})}$
4: while $h' < h$ do
5:     $\mathcal{P} \leftarrow$ append $(e_{cur}, W(e_{cur}), l.t + h')$
6:     if $deg(v_{cur}) = 1$ then
7:         replace the time of $\mathcal{P}$'s last position with $l.t + h$
8:         return $\mathcal{P}$
9:     else if $deg(v_{cur}) = 2$ then
10:        $e_{next} \leftarrow$ an edge incident to $v_{cur}$, except $e_{cur}$
11:    else if $deg(v_{cur}) > 2$ then
12:        if $\mathcal{M}(v_{cur})$ exists then
13:            $e_{next} \leftarrow e_i$ with the highest $\mathcal{M}(v_{cur})[e_{cur}, e_i]$
14:        else if $\mathcal{M}_R(v_{cur})$ exists then
15:            $e_{next} \leftarrow e_i$ with the highest $\mathcal{M}_R(v_{cur})[e_{cur}, e_i]$
16:        else
17:            $e_{next} \leftarrow e_i$ with the most similar travel direction
18:    $v_{cur} \leftarrow$ the other end vertex of $e_{next}$ from $v_{cur}$
19:    $e_{cur} \leftarrow e_{next}$
20:    $h' \leftarrow h' + \frac{W(e_{cur})}{S(e_{cur})}$
21: $\mathcal{P} \leftarrow$ append $(e_{cur}, W(e_{cur}) - S(e_{cur}) \cdot (h' - h), l.t + h)$
22: return $\mathcal{P}$
```

4.3.1 *Comparison with maximum likelihood and PLM*

Maximum Likelihood and PLM keep all candidate (sub)paths produced from the graph traversal until the best path is selected at the times when the algorithms terminate. In contrast, Greedy selects the most probable edge in every iteration and does not consider the other edges (subpath) after the selection. Therefore, Greedy requires less storage and visits much fewer vertices during its graph traversal. This promises high efficiency and scalability for large road network graphs.

Table 5 compares the time complexity of each path prediction method. We also offer practical comparisons among the methods in our experimental study (Fig. 11).

4.3.2 *Example*

Figure 6 illustrates an example of the greedy path prediction method. The query object is currently traveling on e_1 toward v_1. Since a fork dilemma occurs (i.e., $deg(v_1) > 2$), Greedy examines the mobility statistics of v_1, which contains

Table 5 Time complexities of prediction methods

Method	Time Complexity
Greedy	$O(\|V_P\|)$, $\|V_P\|$ is the number of vertices in a path
Maximum likelihood	$O(\|E'\|log\|V'\|)$ with a binary heap for priority queue or $O(\|E'\| + \|V'\|log\|V'\|)$ with a Fibonacci heap, where $\|E'\|(\|V'\|)$ is the number of edges (vertices) within the range that a query object can travel for h
PLM	$O(\|E^*\| + \|V^*\|)$ for depth-first search, as well as $O(\|E\|)$ (or $O(log\|E\|)$ with a spatial index) for exit Point computation, where $\|E\|$ is the total number of Edges, and $\|E^*\|(\|V^*\|)$ is the number of edges (vertices) inside the exit point area. Typically, $\|E^*\|(\|V^*\|) \gg \|E'\|(\|V'\|)$

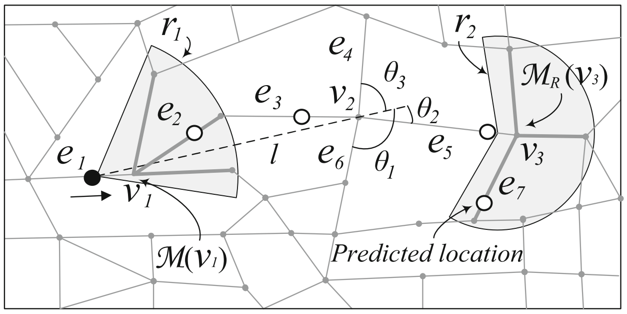


Fig. 6 An example of greedy path prediction

the probabilities for traveling among the next edges (shown in bold edges in the r_1 area of Fig. 6). In this example, $\mathcal{M}(v_1)[e_1, e_2]$ is assumed to have the highest probability, and the prediction process transits to e_2. Next, the method passes over to e_3 (i.e., there is exactly one choice between e_2 and e_3). A fork dilemma then occurs at junction v_2, where no mobility information is available. In this case, we find the edge that has the smallest deviation angle from the travel direction (line l in the figure). Thus, edge e_5 is then chosen as the next predicted edge. After that, the reverse mobility statistics of v_3 are applied to find the most probable travel direction within the r_2 area of the figure because $\mathcal{M}(v_3)$ is unavailable. Eventually, Greedy arrives at e_7 as the final edge in the predicted path.

5 Server-side query processing

In this section, we present a novel indexing method based on the Greedy prediction for supporting efficient processing of predictive range queries on the server side, e.g., "*which objects will be in a given query region 10 min from now*?"

Specifically, we first present the assumed system architecture and the theoretical background for the index; then concrete description of the indexing and query processing follow. Note that existing spatiotemporal indexes such as TPR-trees are specifically designed for the indexing of linear functions and are not applicable here.

5.1 Preliminaries

5.1.1 System architecture

We assume a standard client–server architecture. Each client shares the road network and the Greedy prediction algorithm with the server. To guarantee the same prediction between the clients and the server, they synchronize on the mobility statistics as follows:

For each vertex v with mobility statistics, the client sends its most probable turning pattern (i.e., $\max(\mathcal{M}(v)[e_i, *])$ or $\max(\mathcal{M}_R(v)[e_i, *])$, where e_i is an incident edge of v) to the server. This operation is performed only once, during initialization. The client subsequently updates the mobility statistics on the server only when any of its most probable turning patterns is changed since the last update. In practice, these typical turning patterns for an individual are unlikely to change frequently.

The server stores the reported patterns on vertex v from all clients in its mobility statistics $\mathcal{M}^s(v)[e_i, e_j, O]$, where O is a set of clients, each of which reports $\mathcal{M}(v)[e_i, e_j]$ as the most probable turning pattern from e_i. Therefore, the maximum number of entries on a vertex v is $(deg(v) - 1) \cdot deg(v)$. As the number of objects maintained by the system increases, the server needs to store only larger numbers of objects' identifiers, i.e., O of $\mathcal{M}^s(v)[e_i, e_j, O]$, which can be compressed with various techniques (e.g., a bitmap).

The storage requirement for mobility statistics on the server side is affected by both the spatial network size and the number of objects. We cover the theoretical maximum number of pattern entries and the actual storage consumptions from large amounts of trajectory data in our experiments (Table 8).

5.1.2 Foundation

We first discuss the simple case of considering only one object. Let $h(\mathcal{P})$ be the total travel time of a path $\mathcal{P}$ predicted by Greedy.

Definition 5 Given two vertices v and v', their **prediction distance** $D_P(v, v')$ is defined as:

$$D_P(v, v') = \begin{cases} h(\mathcal{P}) & \textit{if there exists a path } \mathcal{P}: v_1 \succ \cdots \succ v_k \\ & \textit{such that } v = v_1 \textit{ and } v' = v_k \\ \infty & \textit{otherwise} \end{cases} \quad (4)$$

Given network positions p and p' on the edges (v_i, v_j) and $(v_{i'}, v_{j'})$, respectively, we define their *prediction distance* as:

$$D_P(p, p') = \min_{a \in \{i,j\}, b \in \{i',j'\}} \left(\frac{D_L(p, v_a)}{S(v_i, v_j)} + D_P(v_a, v_b) + \frac{D_L(v_b, p')}{S(v_{i'}, v_{j'})} \right).$$

Let C be a grid that partitions the coverage region of a road network into $m \times n$ cells, and let $V(c)$ be a function that returns a set of network positions that are spatially within the cell $c \in C$.

Given two cells $c \in C$ and $c' \in C$, their *prediction distance* $D_P(c, c')$ is defined as:

$$D_P(c, c') = \min_{p \in V(c), p' \in V(c')} D_P(p, p').$$

Lemma 2 is easy to establish based on the above definitions. It will be used as the key filter for our indexing method in the following section. Consider the case where multiple objects contributing to its own value $D_P(c, c')$; only the minimum one is stored at the server. If $D_P(c, c')$ is greater than a given prediction length h, any of the objects currently residing on c cannot reach c' within h. Therefore, the objects on c are safely pruned for identifying the objects that will be in c' after h.

Lemma 2 Prediction distance filter *Given two cells c and c' and a prediction length h, if $D_P(c, c') > h$ then $D_P(p, p') > h$ holds for all $p \in V(c)$, $p' \in V(c')$.*

Likewise, we can adopt the Euclidean distance for pruning objects that cannot travel to c' within h. Let $D_E(c, c')$ be the minimum Euclidean distance between two cells $c \in C$ and $c' \in C$, and let $S_{max}(G)$ be the maximum travel speed among all edges in a graph G. Then, the following lemma holds:

Lemma 3 Euclidean distance filter *Given two cells c and c' and a prediction length h, if $\frac{D_E(c,c')}{S_{max}(G)} > h$, then $D_P(p, p') > h$ holds for all $p \in V(c)$, $p' \in V(c')$.*

Proof The Euclidean distance does not exceed the sum of weights along any path between c and c', and $S_{max}(G)$ is no smaller than any travel speed on an edge along the paths. Thus, $D_P(c, c') \geq \frac{D_E(c,c')}{S_{max}(G)}$. If $\frac{D_E(c,c')}{S_{max}(G)} > h$ then $D_P(c, c') > h$ must hold. Hence, we obtain $D_P(p, p') > h$ by Lemma 2. □

5.2 The prediction distance table

The core idea underlying our indexing technique, named the *prediction distance table*, is to pre-compute the prediction distances between all pairs of cells by performing the Greedy path prediction a priori and then use these distances for pruning objects in an initial filtering step during the query processing.

5.2.1 Index construction and maintenance

In a pre-processing phase, we partition the region covered by the road network into a regular grid C consisting of $m \times n$ cells. We then select a cell $c \in C$ and retrieve all edges that intersect the border of c. For each such edge, we execute Greedy with $h = \infty$ in the direction toward the outside of c until the prediction meets either the network boundary or a dead-end road. This way, we obtain a sequence of cells that Greedy has passed. We record the travel time from one cell to another among the cells obtained from the sequence, while preserving the order of the sequence. We repeat this processing for every cell in C for each object (since each object may have a different Greedy path due to different mobility statistics). We store only the minimum value of the travel times for

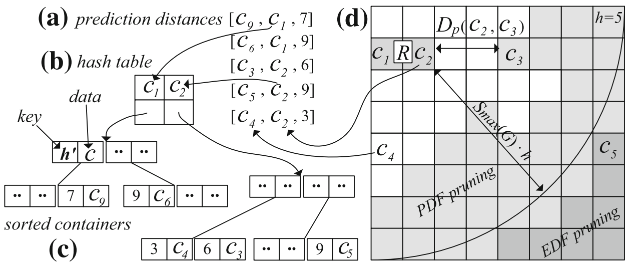


Fig. 7 Storage scheme and a query processing example

each $[c_i, c_j]$, i.e., $D_P(c_i, c_j)$. Therefore, each value for a pair of cells $[c_i, c_j]$ represents the minimum travel time from c_i to c_j among all objects, with respect to the Greedy prediction.

The above process yields a set of index entries of the form $[c_i, c_j, h']$, where $h' = D_P(c_i, c_j)$ (Fig. 7a). We then hash each entry on its destination cell (i.e., c_j) (Fig. 7b). Each data in the hash table points to a B$^+$-tree-like sorted container, where the key value of the container is h', and the data value is the origin cell of the index entry (Fig. 7c).

Theoretically, the number of index entries can reach $m \cdot n(m \cdot n - 1)$, where $m(n)$ is the number of rows (columns) in the data partition. In practice, the number of entries is far below the theoretical bound, for the following reasons:

– Most pairs of adjacent cells have connections (i.e., common edges), meaning that the value of h' for the cells is 0. We index h' in order to be able to prune any entry that has $h' > h$. If h' has value 0, it has no pruning power, so we do not store it.
– In practice, many objects reach only a small part of the network. Due to the locality of prediction paths, not every pair of cells is connected, with respect to the Greedy path prediction. Thus, many empty entries should result from such cells.

It is worth noticing that the size of the index depends only on the granularity of the cells; it is independent of the number of objects.

When a client o reports a new turning pattern $\mathcal{M}(v)[e_i, e_j]$, we perform the pre-computation process only for o, from the edge e_i. We then update the index entry information if any $D_P(c_i, c_j)$ computed for o is smaller than the existing one. When a client reports a change of its most probable turning pattern, we first perform the pre-computation process with old pattern information and remove or update relevant entries if necessary. We then repeat the same process as the new pattern update. As mentioned, turning patterns of an individual typically do not change often, and thus the update operation will be infrequent.

5.2.2 Predictive range query processing

We present two different filtering techniques for processing predictive range queries. The first method, *Euclidean*

Distance Filter (EDF), is based on Lemma 3. The key idea is that it prunes objects that are further away than the Euclidean distance an object can travel during a given prediction length h at the highest travel speed. Specifically, let $S_{max}(G)$ be the maximum travel speed in a road network G. Then the maximum travel distance is $S_{max}(G) \cdot h$. Any objects in cells beyond this boundary can be pruned safely based on Lemma 3 (e.g., the dark gray cells in Fig. 7d).

The second filtering method, *prediction distance filter* (PDF), utilizes the prediction distance table. Based on Lemma 2, we can quickly identify the objects that cannot reach the query cells within a given prediction length. For instance, in Fig. 7d, we first find three candidate cells c_3, c_4, and c_5 that have query cell c_2 as travel destination by accessing the hash table (Fig. 7b). We then scan the sorted container from the beginning and find c_4 since the prediction distance h' of $[c_2, c_4]$ is smaller than the given $h = 5$. However, this search terminates as soon as it meets $h' = 6$ (i.e., $D_P(c_2, c_3)$) because its value exceeds h. Finally, we prune the objects in cells c_3 and c_5 without further inspection. Note that the search bound of PDF is usually much tighter than that of EDF (the dark gray area in Fig. 7d) because EDF uses the fastest speed with the Euclidean distance.

Algorithm 3 outlines the pseudo-code of PDF. Lines 3–4 find the entries whose destination cells belong to the given query window. In Lines 5–7, the algorithm reports the objects within the origin cells as candidates; however, this process runs only for the entries whose prediction distances are smaller than h (Lines 6–7). Finally, we report the objects that are in the query cells and their neighbors (Lines 8–10) because the index does not store prediction distances for neighbor cells.

Algorithm 3 Prediction distance filter (G, O, T, R, h)

```
Input: graph $G$, a set of objects $O$, prediction distance table $T$
    query window $R$, prediction length $h$
Output: a set of candidate objects $O_{cand}$
1: $O_{cand} \leftarrow \emptyset$
2: $Q \leftarrow$ all intersecting cells with $R$
3: for each cell $q \in Q$ do
4:     sorted container $T_q \leftarrow T$.find($q$)
5:     if $T_q$ exists then
6:         for each entry $e \in T_q$ such that $e.h' \leq h$ do
7:             $O_{cand} \leftarrow O_{cand} \cup o \in O$ within $e.originCell$
8: $Q' \leftarrow Q \cup$ neighbor cells of $Q$
9: for each cell $q' \in Q'$ do
10:    $O_{cand} \leftarrow O_{cand} \cup o \in O$ within $q'$
11: return $O_{cand}$
```

6 Experiments

The objective of the experimental study is twofold. First, we compare the performance of client-side prediction among our proposals and competitors, summarized in Table 6. Second, we investigate the performance of the server-side query processing.

We note that both Linear-A* and RMF-A* in Table 6 are built upon prediction models designed for free-space moving objects. Since they may still have some potential for predicting network-constrained moving objects, we cover them in the experimental study.

6.1 Datasets and experimental settings

We experiment with real datasets in order to contend with real world phenomena. We use two real trajectory datasets and their corresponding road networks. Table 7 summarizes their details.

- **Aalborg** [12]: GPS logs from 20 cars collected over several months in Aalborg in a project that investigated driver response to speeding alerts issued by in-car devices.
- **Copenhagen** [11]: Positions from 192 private cars were logged every second over a 1-year period, in a project that studied the feasibility of road pricing in Copenhagen. This substantial dataset contains 114,393 trajectories.
- **Road networks**: Each road segment has a length (i.e., weight) and a speed limit, and each vertex has spatial coordinates.

As defined in Sect. 3, the prediction accuracy of each method can be measured in two ways: (i) the *network distance*

Table 6 Prediction methods for comparison

Category	Method	Description
Our Proposals	MaxLike	Maximum likelihood [Sect. 4.2]
	Greedy	[Sect. 4.3]
Competitors	PLM	Most relevant work [16] to ours
	Linear-A*	Adoption of linear prediction [25]
	RMF-A*	Adoption of non-linear prediction [26]

Table 7 Trajectory datasets and road networks

Dataset name	Aalborg	Copenhagen
Trajectory data		
Objects	20	192
Trajectories	4,542	114,393
Total positions	1,236,324	43,896,907
Road networks		
Edges	45,598	115,467
Vertices	36,125	85,309
Coverages (km)	43.9 × 48.2	54.5 × 43.2

between a predicted location and its corresponding actual location and (ii) the *path similarity* between the edges in the predicted path and those in the corresponding actual query trajectory. The F1-score is used to represent the edge similarity.

The query workload consists of 300 trajectories randomly chosen from the trajectory dataset. For each query trajectory, we take a position as the current location and then perform predictions with different prediction length h. Next, we set the current location to be 10 time units after the last one and repeat the same prediction process. We keep sliding the current location and execute the predictions throughout the query trajectory. The prediction errors are recorded by the two measurements, and we average the errors across the whole query workload. To be fair, we exclude the query workload from the trajectory dataset during the mobility pattern discovery.

Since there are not sufficiently many trajectories that exceed 30 min in Aalborg, we perform the prediction tests using h values up to 10 min for this dataset.

All the prediction methods are implemented in the C++ language, and the prediction experiments are run on a computer with an Intel Xeon 2.4 GHz processor and 4 GB of main memory. Our index, i.e., the prediction distance table, uses a memory-based implementation, as well as all mobility statistics discovered from the datasets are stored in the main memory.

6.2 Client-side prediction performance

In the first set of experiments, we compare the prediction errors of our methods with that of their competitor PLM [16]. In addition, we include two variants of prediction methods for objects in Euclidean space that are based on linear [25] and non-linear [26] prediction models. These variants first predict a location at a given query time and then apply a map-matching algorithm to the predicted location for obtaining a network position. A* shortest path search is applied to compute the predicted path from the current position to the predicted location. We name these methods Linear-A* (linear prediction) and RMF-A* (non-linear model).

Figure 8 shows that the prediction accuracies of our methods (MaxLike and Greedy) exceed those of the competitors. Specifically, the prediction errors of our proposals are around half of those of Linear-A* and RMF-A*, and about one quarter of those of PLM.

Between our methods, MaxLike shows slightly lower errors than does Greedy for short-term predictions ($h <$ 15 min); surprisingly, this trend is reversed for longer-term predictions for Copenhagen. This is because Greedy employs reverse mobility statistics and travel directions for predicting turns at road junctions without mobility statistics. As we will see shortly, utilizing travel directions improves prediction

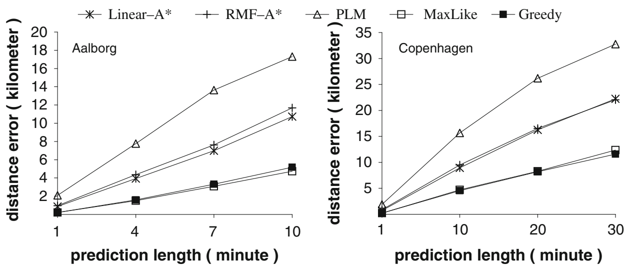


Fig. 8 Comparison of prediction errors

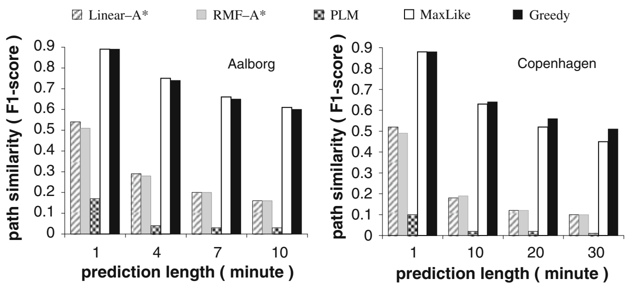


Fig. 9 Comparison of predicted path similarity

accuracy. For long-term predictions, a predicted path may contain many junctions without statistics; hence, the benefits of maximizing probabilities in MaxLike decrease.

Also, Linear-A* exhibits slight higher accuracies than does RMF-A*. The RMF method needs several previous positions to compose its motion function. Although an object usually travels along a relatively straight line (i.e., a road segment) after turning at a road junction, the positions on the previous road still influence the motion function computation for several time points. This degrades the accuracy of RMF-A* for road-network constrained movement.

The better accuracies of our prediction methods are also clear from the path similarity comparisons reported in Fig. 9. For both datasets, the path similarities of our methods decrease gradually with increasing h, while the similarities of the other methods decrease more markedly. For 30-min predictions, Greedy reports at least 5.1 times higher similarity than do Linear-A* and RMF-A*, and Greedy is 49.4 times better than PLM. These differences are much greater than the distance error differences from the previous experiment.

Figure 10 explains well the reasons why the differences in path similarities between our methods and the other methods are bigger than are the corresponding distance errors. The figure illustrates predicted paths obtained by all methods for $h = 4$ min using Aalborg. MaxLike and Greedy predict identical paths in this test. Although the destinations (i.e., the ends of the forecasted paths) predicted by our methods and Linear-A* exhibit similar distance errors (i.e., have similar distances to at the end of the actual trajectory), the path predicted by Linear-A* overlaps little with the actual path when compared with the paths computed by our methods.

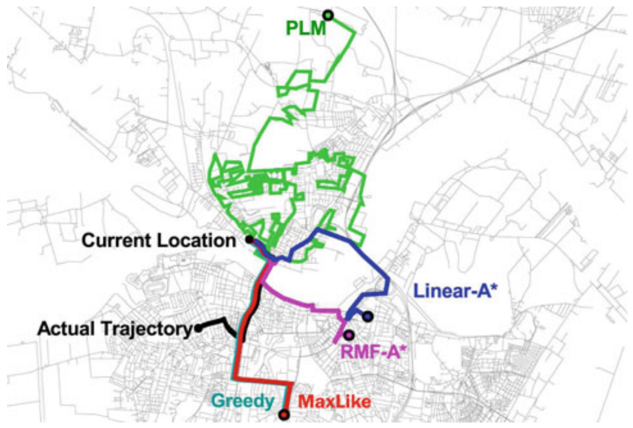


Fig. 10 Visualization of predicted paths (h = 4 min)

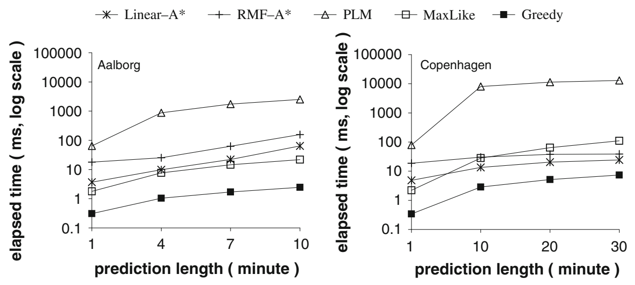


Fig. 11 Comparison of prediction efficiency

Figure 10 also suggests an explanation for the large prediction errors of PLM. Since PLM utilizes depth-first graph search until it finds all exit points (see Sect. 2), the number of edges associated with a path to an exit point may be large, and the path may also be impractical, as shown in the figure.

Next, we also compare the efficiency of prediction for each method in Fig. 11. Greedy exhibits by far the best performance, which is better than those of the competitors by between approximately one order of magnitude and three orders of magnitude. Furthermore, while the processing times of the other methods (especially MaxLike) increase significantly for long-term predictions in Copenhagen, Greedy remains very low (at 7.3 milliseconds for 30-min predictions). It is worth noticing that Greedy also exhibits the best accuracy for the long-term predictions in the first experiment.

Because PLM needs to traverse the argument graph until it finds all exit points, its efficiency decreases significantly as the prediction length increases. The query processing times of Linear-A* and RMF-A* also increase with growing h because the search space in the graphs for the A* algorithm is increased as the predicted locations are moved further from the current position.

Our path prediction methods utilize past movement patterns (i.e., mobility statistics at road junctions and mined speeds of road segments) for path prediction. It is of interest to see how the availability of movement patterns affects prediction.

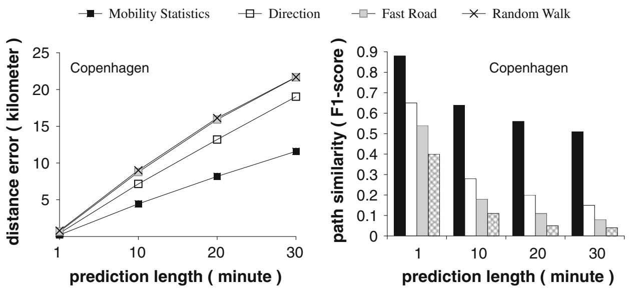


Fig. 12 Effect of mobility pattern use for prediction

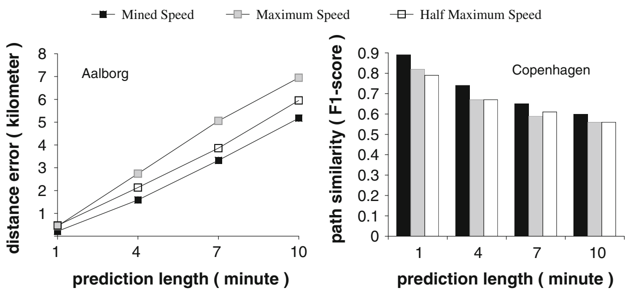


Fig. 13 Effect of different speed use for prediction

Figure 12 compares the prediction accuracies obtained by using Greedy with different way of choosing the next edge for prediction, on the Copenhagen dataset. *Random Walk* selects the next edge randomly when encountering a fork dilemma. *Fast Road* chooses the edge with the highest speed limit as the next edge, whereas *Direction* selects the edge with the most similar angle to the object's current travel direction as the next edge. The experiment using the Aalborg dataset yields similar trends, so the results are omitted. Clearly, using mobility statistics improves the prediction performance greatly; the distance errors are approximately half of those of Fast Road and Random Walk. Although Direction is an improvement over the latter two, its error also increases substantially with increasing h; in contrast, Greedy with mobility patterns is affected less by growing h. Furthermore, the use of mobility statistics outperforms the others in terms of the path similarity.

MaxLike assigns equal probability to each next edge when mobility statistics unavailable, which resembles the approach of Random Walk. In contrast, Greedy uses the approach of Direction for selecting the next edge in this case. As Direction is better than Random Walk, this difference gives Greedy an advantage over MaxLike. Both Greedy and MaxLike estimate an object's travel time on a road segment based on speeds mined from historical trajectories. The use of mined speed for prediction is investigated in Fig. 13. The results for Maximum Speed and Half Maximum Speed are measured by applying the maximum speeds and half of the maximum speeds of road segments for travel time estimation.

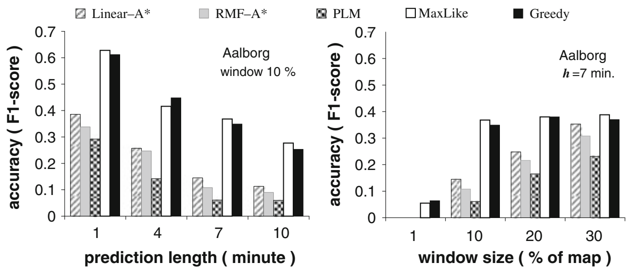


Fig. 14 Accuracy of predictive window query

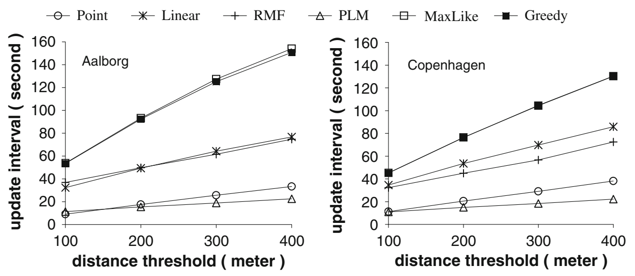


Fig. 15 Location update frequencies

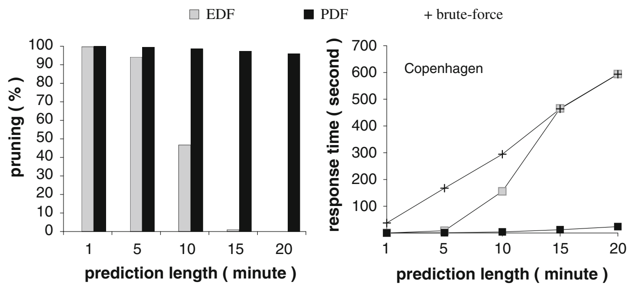


Fig. 16 Pruning power and query response time

In comparison with the result of the previous experiments, the path similarities are not sensitive to the use of the different speeds. This is because the speeds do not affect the selection of the next edge, but only how far an object travels within a given prediction length.

6.3 Server-side prediction performance

For the server side, we first investigate the prediction accuracy of predictive window queries. For this experiment, we generate a category of windows for each of 4 sizes, with each category containing 100 windows placed randomly within the road network. Next, we take all trajectories in the dataset as query inputs. For each trajectory, we pick a random position as the current location and examine whether the position h time units after the current location is within any of the windows generated. We also perform prediction using each method with the same current location and record whether a predicted location is covered by any of the windows. By doing this for all the trajectories, we obtain an actual set of objects and a predicted set of objects (i.e., S_{act} and S_{pred}, respectively, in Sect. 3) for each query window. Finally, the F1-score is computed for each window, and we report averages over all scores.

Figure 14 presents the accuracy results for predictive window queries for each method, for different prediction lengths and window sizes. The results for Copenhagen are similar and are thus omitted. Although the accuracies of MaxLike and Greedy decrease with growing h, they are always better than the other methods. This is evidence that a more accurate prediction model increases the quality of query results. When varying the window size, we observe that our methods perform better than the others with smaller windows. For very large windows (30% of the network coverage), the differences among the methods are small, and linear prediction becomes competitive. This is so because the probability that even an inaccurately predicted location is within a query window is high if the window occupies a significant fraction of the entire space.

We next examine how the prediction accuracy affects the client–server location updates. In Sect. 3, we observed that a low update frequency is important for good system performance, and we suggested that accurate predictions will reduce the update rate. Figure 15, which plots average time durations in-between consecutive updates for different distance threshold values, offers quantitative evidence that substantiates this claim (the results for MaxLike and Greedy intersect for Copenhagen, which hides the MaxLike results). In the figure, 'Point' denotes point-based location tracking [8], in which an update is performed when the distance between the most recently reported position and the current position exceeds the given distance threshold value. The update frequencies of MaxLike and Greedy decrease substantially with increasing distance thresholds, while the frequencies of the other methods decrease less markedly. The improved predictions reduce not only the overhead of server-side updates, but also the mobile communication costs.

In the following experiments, we analyze the effect of our indexing method on the processing of predictive range queries. We use the Copenhagen dataset to study scalability. We regard each trajectory as coming from a distinct object (i.e., the number of objects equals 114,393) and also use the first position of the trajectory as the object's current location. The size of each grid is set to 5 times the average length of an edge (462 m), and 10 query windows are generated randomly within the network; each query window size is fixed at 10% of the network coverage. Under these settings, we obtain 1,510,658 index entries (18 MBytes) with a 43-min construction time.

Figure 16 characterizes the pruning power of our indexing method, and it plots the query response time (CPU time) of predictive range queries, based on the index. The results

Table 8 Overview of discovered patterns

	Aalborg		Copenhagen	
	Bound	Having patterns	Bound	Having patterns
Vertices	27,268	6,059 (22.2%)	71,015	35,887 (50.5%)
Objects	20	2.8 (14.0%)	192	15.3 (8.0%)
Entries	2,277,500	29,890 (1.3%)	59,400,192	974,866 (1.6%)
Speed	91,196	11,150 (12.2%)	230,934	90,043 (38.9%)

for EDF are obtained by pruning objects that are further away than the Euclidean distance an object can travel during h time units at the highest mined speed. The results for PDF are obtained by utilizing the prediction distance table (see Sect. 5.2) for filtering. Observe that EDF is only useful for near-future predictions; its pruning power diminishes for longer than 10-min predictions. In the Copenhagen trajectory dataset, the highest speed mined was 144 km/h. As a result, the bound used for EDF quickly exceeds the road network, resulting in no filtering.

In contrast, PDF's pruning power remains high (96.1%) even for $h = 20$ min because PDF is based on a prediction space whose distances are much greater than those used in EDF. In addition, the query response time of PDF remains very low even for long prediction durations. In the tests, we store the objects' current locations in the main memory. If the objects' locations were stored on disk, the differences in query response time between PDF and the brute-force method would be much greater.

Next, we study the storage consumption of the patterns reported by all clients to the server, for the server-side Greedy prediction. Table 8 lists (i) the numbers of entries that may have mobility statistics and (ii) the numbers of edges that have mined speeds. In order to obtain the largest numbers for the patterns, we set the values of minimum support and minimum confidence (see Sect. 3) for the discovery of mobility statistics to 1 and 0.0, respectively. (These setting are also applied for the other experiments covered in this paper.) In the table, 'bound' captures the maximum values for the different patterns. The numbers for 'vertices' are the numbers of vertices with a fork dilemma and thus differ from the total numbers of vertices in the road networks. Next, 'objects' describe the average numbers of objects on the vertices having mobility statistics. The bounds for 'speed' are two times the numbers of edges in the road networks. Since we mine travel speeds according to two different time categories, i.e., traffic time and the rest, each edge has at most two mined speeds associated with the time categories.

It follows from the table that the movement patterns discovered are relatively sparse. This is because people usually drive in certain areas, often following the same routes, and do not often visit new locations. For instance, although the size of Copenhagen is very large, relatively few mobility entries are mined. Moreover, by setting the threshold values for the pattern discovery (i.e., minimum support and minimum confidence) slightly higher, we can further reduce the pattern sizes substantially, while preserving strong patterns.

7 Conclusions

Most existing object movement prediction schemes focus on near-term predictions, e.g., to reduce update rates. These techniques are incapable of predicting the turning behaviors of moving objects at road junctions.

To address this, we develop a *network mobility model* for effectively and concisely capturing the turning patterns of moving objects at road junctions and estimating the objects' travel speeds on road segments. Based on this model, we develop two algorithms for predicting the future path of a mobile user moving in a road network. The *Maximum Likelihood* algorithm returns paths that maximize the travel probability among all possible paths, while the *Greedy* algorithm aims at being highly efficient while computing near-maximum probability paths. Furthermore, we present a novel indexing method that utilizes the Greedy algorithm for supporting efficient processing of predictive range queries on the server side.

Extensive experiments with real data suggest that both algorithms outperform existing prediction solutions for objects moving in a road network. They also confirm that our indexing technique efficiently processes predictive range queries on the server side.

Acknowledgements Christian S. Jensen is an Adjunct Professor at University of Agder, Norway.

References

1. Aggarwal, C.C., Agrawal, D.: On nearest neighbor indexing of nonlinear trajectories. In PODS, pp. 252–259 (2003)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB, pp. 487–499 (1994)

3. Anagnostopoulos, T., Anagnostopoulos, C.B., Hadjiefthymiades, S., Kalousis, A., Kyriakakos, M.: Path prediction through data mining. In: International Conference on Pervasive Services, pp. 128–135 (2007)
4. Baeza-Yates, R.A., Ribeiro-Neto, B.A.: Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc. (1999)
5. Brilingaitė, A.: Location-related context in mobile services. PhD in Computer science, Aalborg University (2006)
6. Brilingaitė, A., Jensen, C.S.: Online route prediction for automotive applications. In: World Congress and Exhibition on Intelligent Transport Systems and Services (2006)
7. Cho, H., Chung, C.: An efficient and scalable approach to CNN queries in a road network. In: VLDB, pp. 865–876 (2005)
8. Čivilis, A., Jensen, C.S., Pakalnis, S.: Techniques for efficient road-network-based tracking of moving objects. TKDE **17**(5), 698–712 (2005)
9. Ding, Z., Güting, R.H.: Managing moving objects on dynamic transportation networks. In: SSDBM, pp. 287–296 (2004)
10. González, M.C., Hidalgo, C.A., Barabási, A.: Understanding individual human mobility patterns. Nature **453**(7196), 779–782 (2008)
11. Jensen, A.B.O., Zabic, M., Overø, H.M., Ravn, B., Nielsen, O.A.: Availability of GNSS for road pricing in copenhagen. In: GNSS, pp. 2951–2961 (2005)
12. Jensen, C.S., Lahrmann, H., Pakalnis, S., Runge, J.: The infati data. CoRR, cs.DB/0410001, (2004)
13. Jensen, C.S., Lin, D., Ooi, B.C.: Query and update efficient B+-tree based indexing of moving objects. In: VLDB, pp. 768–779 (2004)
14. Jensen, C.S., Lin, D., Ooi, B.C., Zhang, R.: Effective density queries on continuously moving objects. In: ICDE, pp. 71 (2006)
15. Jeung, H., Liu, Q., Shen, H.T., Zhou, X.: A hybrid prediction model for moving objects. In: ICDE, pp. 70–79 (2008)
16. Karimi, H.A., Liu, X.: A predictive location model for location-based services. In: ACM GIS, pp. 126–133 (2003)
17. Kim, S., Won, J., Kim, J., Shin, M., Lee, J., Kim, H.: Path prediction of moving objects on road networks through analyzing past trajectories. In: KES, pp. 379-389 (2007)
18. Kollios, G., Gunopulos, D., Tsotras, V.J.: On indexing mobile objects. In: PODS, pp. 261–272 (1999)
19. Liu, T., Bahl, P., Chlamtac, I.: Mobility modeling, location tracking, and trajectory prediction in wireless atm networks. IEEE J. Sel. Areas Commun. **16**(6), 922–936 (1998)
20. Liu, X., Karimi, H.A.: Location awareness through trajectory prediction. Comput. Environ. Urban. Syst. **30**(6), 741–756 (2006)
21. Mokbel, M.F., Xiong, X., Aref, W.G.: Sina: scalable incremental processing of continuous queries in spatio-temporal databases. In: SIGMOD, pp. 623–634 (2004)
22. Patel, J.M., Chen, Y., Chakka, V.P.: STRIPES: an efficient index for predicted trajectories. In: SIGMOD, pp. 635–646 (2004)
23. Pathirana, P.N., Savkin, A.V., Jha, S.: Location estimation and trajectory prediction for cellular networks with mobile base stations. IEEE Trans. Veh. Technol. **53**(6), 1903–1913 (2004)
24. Šaltenis, S. Jensen, C.S.: Indexing of moving objects for location-based services. In: ICDE, pp. 463–472 (2002)
25. Šaltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the positions of continuously moving objects. In: SIGMOD, pp. 331–342 (2000)
26. Tao, Y., Faloutsos, C., Papadias, D., Liu, B.: Prediction and indexing of moving objects with unknown motion patterns. In: SIGMOD, pp. 611–622 (2004)
27. Tao, Y., Papadias, D.: Time-parameterized queries in spatio-temporal databases. In: SIGMOD, pp. 334–345 (2002)
28. Tao, Y., Papadias, D., Sun, J.: The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In: VLDB, pp. 790–801 (2003)
29. Wolfson, O., Yin, H.: Accuracy and resource concumption in tracking and location prediction. In: SSTD, pp. 325–343 (2003)
30. Yasdi, R.: Prediction of road traffic using a neural network approach. Neural Comput. Appl. **8**(2), 135–142 (1999)
31. Yiu, M.L., Tao, Y., Mamoulis, N.: The bdual-tree: indexing moving objects by space filling curves in the dual space. VLDB J. **17**(3), 379–400 (2008)