Similarity-Based Prediction of Travel Times for Vehicles Traveling on Known Routes*

Dalia Tiesyte
dalia@cs.aau.dk

Christian S. Jensen
csj@cs.aau.dk

Department of Computer Science
Aalborg University, Denmark

ABSTRACT

The use of centralized, real-time position tracking is proliferating in the areas of logistics and public transportation. Real-time positions can be used to provide up-to-date information to a variety of users, and they can also be accumulated for uses in subsequent data analyses. In particular, historical data in combination with real-time data may be used to predict the future travel times of vehicles more accurately, thus improving the experience of the users who rely on such information. We propose a Nearest-Neighbor Trajectory (NNT) technique that identifies the historical trajectory that is the most similar to the current, partial trajectory of a vehicle. The historical trajectory is then used for predicting the future movement of the vehicle. The paper's specific contributions are two-fold. First, we define distance measures and a notion of nearest neighbor that are specific to trajectories of vehicles that travel along known routes. In empirical studies with real data from buses, we evaluate how well the proposed distance functions are capable of predicting future vehicle movements. Second, we propose a main-memory index structure that enables incremental similarity search and that is capable of supporting varying-length nearest neighbor queries.

1. INTRODUCTION

Geographical positioning and wireless communication technologies enable centralized, continuous position tracking of moving vehicles. A number of applications in the areas of logistics, transit services, cargo delivery, and collective transport involve the management of fleets of vehicles that are expected to travel along known routes according to fixed schedules. Positions accumulated during this process form vehicle trajectories. A trajectory is a function that, given a time, returns the position of a vehicle as it traverses its route. Such trajectories are useful for subsequent analysis and for predicting more accurately future travel times, thus improving user experience and reducing operational costs. Commercial systems already exist that manage vehicle positions in real time and predict the future movements of the vehicles. For example, the public buses in the Aalborg region (Denmark) are equipped with PCs, GPRS-based connectivity to a central server, and GPS devices for positioning. The system also includes on-line displays at the central terminal and major bus stops that provide real-time bus arrival information to the passengers. While methods for travel time prediction for buses are available today (discussed in more detail in Section 2), more accurate and general prediction is needed that is easy to integrate into various existing systems. The prediction accuracy is of interest not only to passengers, but also to the carriers. In some countries (including Denmark and the UK), the carriers are penalized if the bus arrival times that are provided to the passengers are not close to the actual times.

The travel times of vehicles traveling on public roads are influenced by a number of hard-to-predict factors (e.g., traffic accidents, weather conditions, unscheduled events) as well as factors that are known in advance (e.g., the time of the day, the day of the week, holidays, major public events). It is infeasible to enumerate all such factors, but it may also not be necessary. We believe that by using similar trajectories from the past, in many cases it is possible to predict the future travel of a vehicle en route with good accuracy. Specifically, as the vehicle moves along its route, it is possible to find the past trajectory that most resembles the travel by the vehicle so far, denoted the Nearest Neighbor Trajectory (NNT). The future part of this trajectory is then used for predicting the future travel behavior of the vehicle.

The use of historical trajectories for real-time prediction of vehicle movement requires a notion of similarity (or distance) between vehicle trajectories. A number of authors (discussed in Section 2) propose distance measures for time series and two-dimensional trajectories, where the main focus is on spatial similarity. In our case, the requirements are different:

- Since the vehicles are traveling on known routes, their movement can be considered as one-dimensional: the position of a vehicle is defined as the distance it has traveled from the start of its route.
- The trajectories to be compared are expected to have the same lengths. The samples that make up the trajectories are taken at fixed locations, called timing points, along a route; times for some timing points may be missing.
- Since the timing points are fixed, it makes little sense to stretch, shift, or otherwise transform a trajectory. It is only of interest to compare travel patterns on exactly the same segments of a route.
- The similarity measure should allow to choose such trajec-

*This work was funded in part by the Danish Research Agency's Programme Commission on Nanoscience, Biotechnology, and IT. The early stages of this work were performed at School of Computing, National University of Singapore, where the guidance of Profs. Mong-Li Lee and Wynne Hsu was invaluable. The authors also thank Man Lung Yiu for excellent ideas and fruitful discussions.

ACM GIS '08, November 5-7, 2008. Irvine, CA, USA
(c) 2008 ACM ISBN 978-1-60558-323-5/08/11 ...$5.00.

tory that is expected to give the best prediction for the future, even though the past may differ at some points.

As a vehicle proceeds along its route, the NNT-based prediction of its movement has to be revised periodically—it is possible that the actual movement has deviated from the prediction, making it appropriate to form a new prediction. Having available a large amount of historical trajectories is expected to result in more accurate predictions. As the real-time trajectory has to be compared to all historical trajectories, it may be beneficial to use a data access structure that enables pruning of some of the historical trajectories. The requirements differ from those associated with the usual indexing of multi-dimensional data or time series. First, the query length is not known in advance. Even if its length is fixed to l, overlapping indexes have to be created for each sub-sequence of length l. Furthermore, the real-time trajectory grows as the vehicle proceeds. We thus expect that an incremental search algorithm may improve the average query performance.

The problem addressed in this paper is two-fold. First, we analyze possible distance measures for trajectories of vehicle moving on known routes. More specifically, we retrieve the NNT among the historical trajectories, using the available, partial real-time trajectory from the start of the route until the most recently visited timing point. The evaluation criterion is the prediction accuracy. The near-future travel times may be of higher interest than later ones; therefore, the system enables the prioritization of the prediction accuracies of those. Second, efficient, real-time access to similar trajectories is needed. We propose a main-memory index that enables the retrieval of trajectories similar to a partial, real-time trajectory and also enables updates to the query result as times for more timing points are received for the real-time trajectory. The index extends an existing approach to be incremental [10].

The remainder of this paper is organized as follows. Section 2 covers related work in the areas of similarity search and vehicle travel time prediction. Section 3 defines the terms to be used throughout the paper and also the problem to be solved. Section 4 analyzes the distance measures and travel time prediction using those, while Section 5 describes the data structure and algorithms for similarity search. Section 6 presents experimental results. Finally, Section 7 concludes.

2. RELATED WORK

Considerable research has been conducted on similarity-based retrieval of one-dimensional time series, including sensor data, e.g., [3], as well as 2-dimensional trajectories of moving objects. Vlachos et al. [24] study the retrieval of longest common subsequences from vehicle trajectories, and they use a hierarchical clustering tree for data access. They compare only spatial coordinates, excluding the temporal component. Chen et al. [5, 6] use edit distance with extensions (EDR). They propose pruning based on the combination of Q-grams, a table of pre-calculated EDR distances, and frequency histograms. Pekelis et al. [19] propose a number of distance measures for spatial, spatio-temporal, and speed-spatial similarity for the purpose of clustering. In their experiments, their measures outperform the EDR distance when used for clustering. Bakalov et al. [1] group spatio-temporal trajectories into bounding *envelopes* and further into an R-tree-like structure. Commonly used measures, such as Dynamic Time Warping (DTW) that was first proposed for speech recognition [21], involve shifting and stretching and are therefore not suitable for the trajectories considered in this paper. When the search space is restricted, as it is in the problem discussed here, it is desirable to use those restrictions to obtain a faster performance and a more precise match.

Hwang et al. [12] propose a similarity measure for spatio-temporal trajectories of vehicles. Similar trajectories are required to traverse the same *points of interest* (POIs). The L^p-norm is measured in-between arrival times to the POIs. Subsequent work by the authors [13] also introduces TOIs, times of interest, which is a general concept used to define peak times, weekends, etc. The trajectories we consider are different in that only the relative temporal component is of interest, while the positions (timing points) are fixed according to pre-defined routes. Furthermore, the similarity measure has to be predictive: trajectories that are similar in the past are expected to be similar in the future. To the best of our knowledge, similarity search in our context of collective transport has not been discussed yet.

Real-time prediction of bus arrival times has been studied for a couple of decades, and automated vehicle location (AVL) systems is an active research area within intelligent transport systems. Bus arrival-time prediction is an important part of an AVL system. Commonly used prediction methods include Kalman filtering (KF), e.g., [4, 9, 22], and artificial neural networks (ANNs), e.g., [7, 11, 14, 15, 18]. Other machine learning methods such as support vector machines [2] are also applicable.

A model that utilizes historical data and current conditions (including weather) has been proposed [8]. Some researchers use simple algorithms, e.g., prediction according to deviation from a schedule [17]. A prediction model that includes detection of routes and continuous queries on future arrival times has also been proposed [20]. KF prediction is mainly based on real-time data, although historical data can also be used as an external influence parameter. KF works well when only short-term prediction is needed, but deteriorates when prediction is needed for more than one time step. ANNs and other learning methods are based on historical data. Their learning is computationally expensive; therefore, ANNs cannot be updated in real time. In contrast, our proposed arrival-time prediction technique aims to exploit all the historical data available, and it does not require any external data, such as real-time traffic information, which promises easier integration into existing transport management systems.

3. PROBLEM DEFINITION

The problem discussed in this paper is two-fold. First, a distance measure has to be found that allows the prediction of the future travel times of a vehicle based on historical data. Second, efficient access to historical trajectories is needed.

3.1 Functions and Data Representation

A vehicle's geometrical route in two-dimensional space is represented in a one-dimensional space using linear referencing, where the position of each point is the distance from the start of the route. A number of points of interest, termed *timing points*, are chosen, and are used for the representation of both routes and trajectories. In the area of collective transport, a natural choice of such points is the bus stops, since it is of interest to predict the arrival times for those. Two subsequent time samples for the same timing point on a route correspond to the arrival to and the departure from that point.

Def. 1. A route R is represented as a sequence of points, $R = \langle p_0, ..., p_n \rangle$, where each point p_i is termed a *timing point*. The points belong to a metric space P, and the value of p_i denotes its total distance along the route from the start of the route; thus $p_i \leq p_{i+1}$.

A raw trajectory is formed straightforwardly by measuring the vehicle's arrival at each timing point in R. Depending on the track-

ing policy used, position updates from a vehicle may, however, arrive at different positions than those in R. The movement prediction function used during the tracking can be used together with the position updates to calculate the approximate travel times -between the timing points. Issues related to trajectory representation are covered elsewhere [23]. Here, it is assumed that a time sample is available for each timing point, i.e., $\langle p_i, t_i \rangle$ is available for all i, whether obtained directly or by approximation.

Def. 2. A raw trajectory function $f_{\mathrm{raw}} : T \rightarrow R$ represents a vehicle's position $p \in R$ at time $t \in T$, and $f_{\mathrm{raw}}(0) = 0$.

Function f_{raw} is a bijection: it is a non-decreasing, monotonous function. The inverse function is $f_{\mathrm{raw}}^{-1} : R \rightarrow T$, where the time $t \in T$ is the travel time from the start of the route to the position $p \in R$. From the above, a new function f_{tr} is derived by setting the time Δt to be the travel time between two subsequent points p_i and p_{i+1}. This representation is used by the distance measures covered in Section 4. Measuring the travel times instead of speed patterns enables the comparison of waiting times as well, where $p_i = p_{i+1}$.

Def. 3. The trajectory function $f_{\mathrm{tr}} : R \rightarrow \Delta T \in F_{\mathrm{tr}}$ defines the travel times between positions such that $f_{\mathrm{tr}}(p_i) = \Delta t_i$ is the travel time between p_{i-1} and p_i. A trajectory is then represented as a sequence of points $\langle p_i, \Delta t_i \rangle$, where some or all of the time values Δt_i may be missing. On a fixed route, a trajectory is compactly represented as a sequence $\langle \Delta t_0, ..., \Delta t_n \rangle$, where $\Delta t_0 = 0$.

In some cases, it is necessary to refer only to a part of a trajectory, denoted as a sub-trajectory, or a *segment*.

Def. 4. The part of a trajectory f_{tr} that is delimited by points p_i and p_j ($i \leq j$) inclusively, is denoted as $f_{\mathrm{tr}}[p_i, p_j]$ and is termed a segment.

When it is clear from the context, the limits $[p_i, p_j]$ are omitted, and the trajectory segment is simply denoted as f_{tr}. When a real-time trajectory f_{tr} is considered, the segment $f_{\mathrm{tr}}[p_0, p_{\mathrm{cur}}]$ is assumed, where p_{cur} is the most recent timing point with a know time sample.

The distance measure for trajectories is closely related to the representation of the trajectories. Trajectory function f_{tr} can be viewed as an n-dimensional point. It is desirable that the distance measure be a metric, although this is not necessary. In general, pruning by distance requires that at least in some cases, it can be inferred that "if I am far away from you, I am also far away from your neighbors." This enables comparison of only some pairs of trajectories in the given dataset, thus enabling pruning.

The following requirements exist for a distance measure:

- A delay during a segment should not cause a penalty during subsequent timing points along the route.
- The measure should apply to subsequences of trajectories.
- The measure should compare only pairs of travel times in-between the same positions of each trajectory.
- The measure should allow to compare trajectories with missing times for some timing points.
- The measure should to some extent tolerate outliers (e.g., incorrect data).

We proceed to define a general distance measure. Specific distance measures are introduced in Section 4.

Def. 5. A distance measure is a function $d : F_{\mathrm{tr}} \times F_{\mathrm{tr}} \rightarrow R^{+} \cup 0$ that expresses the similarity between a real-time trajectory $f_{\mathrm{tr}} = \langle \Delta t_1, ..., \Delta t_n \rangle$ and a historical trajectory $f'_{\mathrm{tr}} = \langle \Delta t'_1, ..., \Delta t'_n \rangle$. A smaller value means that the argument trajectories are more similar.

The *optimal* nearest neighbor of the trajectory f_{tr} is a trajectory f_{onn} that minimizes the prediction error for the future travel times of the trajectory f_{tr}. Since the future is not known at the current time, the f_{onn} is only known after all travel times have been measured.

Def. 6. Given the current position p_{cur}, the trajectory f_{onn} is defined as the trajectory from the set of historical trajectories F_{tr} that minimizes the prediction error err (to be defined shortly) with respect to the future segment of the query trajectory f_{tr}:

$$f_{\mathrm{onn}} = \underset{f'_{\mathrm{tr}} \in F_{\mathrm{tr}}}{\operatorname{argmin}} \, err(f_{\mathrm{tr}}[p_{cur+1}, p_n], f'_{\mathrm{tr}}[p_{cur+1}, p_n]) \ .$$

When more than one f_{onn} candidate is available, ties are broken arbitrarily. The distance $d(f_{\mathrm{tr}}[p_{cur+1}, p_n], f_{\mathrm{onn}}[p_{cur+1}, p_n])$ is called the optimal distance.

At the current time t_{cur}, the optimal nearest neighbor is only hypothetical. The *actual* nearest neighbor of the trajectory f_{tr} is the trajectory f_{nn} that minimizes the distance for the past segment trajectory f_{tr}. It is expected that the actual nearest neighbor will yield a distance that is close to the optimal when it is used in place of f_{onn}.

Def. 7. Given a current position p_{cur} and a distance measure d, the nearest neighbor of a query trajectory f_{tr} is a trajectory $f_{\mathrm{nn}} \in F_{\mathrm{tr}}$ that satisfies the following:

$$f_{\mathrm{nn}} = \underset{f'_{\mathrm{tr}} \in F_{\mathrm{tr}}}{\operatorname{argmin}} \, d(f_{\mathrm{tr}}[p_0, p_{cur}], f'_{\mathrm{tr}}[p_0, p_{cur}]) \ .$$

As before, ties are broken arbitrarily.

Finally, a vector that prioritizes the up-coming timing points with respect to the prediction accuracy is needed. The points that are more important are given higher weights, while the less important points are given lower weights.

Def. 8. A *priority vector* $\overline{PV} = \langle k_{cur+1}, ..., k_n \rangle$ defines the importance of prediction accuracy for each future timing point. A coefficient $k_i \in [0, 1]$ indicates a higher priority when it is closer to 1. The value 0 means that prediction for the associated timing point is not of interest. If $\overline{PV}$ is not given, $k_i = 1$ for all i is assumed.

3.2 Problem Statement

3.2.1 The Prediction Problem

Given a route $R = \langle p_0, ..., p_n \rangle$, a set of historical trajectories F_{tr}, and a real-time sub-trajectory $f_{\mathrm{tr}}[p_0, p_{cur}]$ on route R ($cur < n - 1$), a trajectory $f_{\mathrm{nn}} \in F_{\mathrm{tr}}$ has to be found that gives the most accurate prediction for $f_{\mathrm{tr}}[p_{cur+1}, p_n]$ with respect to the priority vector $\overline{PV}$.

The notion of "the most accurate prediction" still needs definition. The prediction error err of the prediction $f_{\mathrm{pr}} = \langle \Delta t'_{cur+1}, ..., t'_n \rangle$ with respect to the future query trajectory $f_{\mathrm{tr}} = \langle t_{cur+1}, ..., n \rangle$ is a weighted L^p distance, divided by the number of points where $k_i \neq 0$. Higher values of parameter p suggest a preference for lower error variance, while lower values of parameter p suggest a preference for a smaller average error, while some outliers can be tolerated. The error is defined as follows:

$$err(f_{\mathrm{tr}}, f_{\mathrm{pr}}, \overline{PV}) = \frac{1}{\sum_{k_i \neq 0} 1} \sum_{i=cur+1}^{n} k_i (\Delta t_i - \Delta t'_i)^p \ .$$

The trajectory that gives the minimum error err is the optimal nearest neighbor f_{onn} with respect to function err. The problem defined above is, however, unsolvable before the actual travel times $\langle \Delta t_{cur+1}, ..., \Delta t_n \rangle$ are known. Therefore, heuristics are needed. To make the problem solvable, it is redefined as follows.

Given a set of historical trajectories F_{tr} for route R, a distance measure d has to be found such that f_{pr} as defined by Definition 7 would give a small error $err(f_{\mathrm{tr}}[p_{cur+1}, p_n], f_{\mathrm{pr}}, \overline{PV})$.

3.2.2 The Search Problem

Given a real-time trajectory f_{tr}, an NNT has to be retrieved efficiently from the historical dataset F_{tr}. The length of the query is variable, and different distance measures must be accommodated. The aim is to propose a main-memory data structure and associated algorithms that reduce the search time when compared to a sequential scan (SS), and that return correct results.

Given a route R and an associated query trajectory f_{tr} with an arbitrary number of points, a threshold thr_{d}, and a distance measure d, the trajectory f_{nn} that minimizes d (Definition 7) has to be retrieved. The approximation threshold thr_{d} restricts f_{nn} to be no further than $d_{\min} + thr_{\mathrm{d}}$ from the query, where $d_{\min}$ is the distance to the actual NNT.

4. DISTANCE MEASURES

In this section we consider specific distance measures that are appropriate for our problem setting. Many measures that are commonly used for comparison of time series, including Dynamic Time Warping, are not appropriate: when the routes are fixed, exact positions have to be matched, and trajectory transformation (shifting or stretching) is not allowed.

4.1 Longest Common Subsequence

Vlachos et al. [24] propose similar trajectory search based on a Longest Common Subsequence (LCSS) measure. With the help of hierarchical clustering, a tree is built and then used for the efficient access to similar trajectories. Our setting with one-dimensional routes makes it possible to simplify the measure because the corresponding subsequences of two trajectories must be matched. Also, non-contiguous subsequences are possible.

Given the threshold thr, the distance $LCSS_{\mathrm{thr}}$ between two trajectories f_{tr} and f'_{tr} is:

$$LCSS_{\mathrm{thr}}(f_{\mathrm{tr}}, f'_{\mathrm{tr}}) = \frac{\sum_{|\Delta t_i - \Delta t'_i| \leq thr} 1}{l} \ ,$$

where $\Delta t_i \in f_{\mathrm{tr}}$, $\Delta t'_i \in f'_{\mathrm{tr}}$, and l is the number of points used in the query. The threshold thr is introduced to reduce unpredictable ("white") noise. The measure is robust to the presence of a small amount of outliers; however, it is important to choose the right thr that filters outliers and still captures the near trajectories.

4.2 L^p Distance

Each point in the trajectory f_{tr} can be viewed as a point in an n-dimensional L^p-norm space. The most common variant of the L^p-norm is the Euclidean distance (L^2-norm). This measure is a metric, and its computation is efficient. Given the representation of a trajectory as in Definition 3, the delays at some segments in a trajectory do not influence the error penalty at other segments. The higher the p value, the less tolerable are outliers. Given that the trajectories are of equal length, the L^p-norm does not cause the problem of choosing which segments correspond to which during the matching. We use the L^p-norm distance to the power of p, divided by the query length l:

$$L^p(f_{\mathrm{tr}}, f'_{\mathrm{tr}}) = \sum_{i=cur-l+1}^{cur} \frac{1}{l} (\Delta t_i - \Delta t'_i)^p \ .$$

4.3 Other Measures

A number of distance measures for one-dimensional sequences exist that might be applicable for trajectories when considering these as n-dimensional vectors. A correlation-based distance $d_\beta = \left(\frac{1-\rho}{1+\rho}\right)^\beta$, where ρ is Pearson's correlation coefficient, considers the trajectories similar when they are proportional to each other. When using this measure, the times, predicted by f_{nn}, should be multiplied by the coefficient of the proportion.

The Chebyshev distance d_{ch} is the maximum distance between two points in the trajectories: $d_{ch} = \max |\Delta t_i - \Delta t'_i|$. This measure does not tolerate any outliers. It is an extreme case of the L^p-norm, where $p \to \infty$.

The Canberra distance is given by $d_{ca} = \sum_i \frac{|\Delta t_i - \Delta t'_i|}{\Delta t_i + \Delta t'_i}$. The point 0 is the furthest point when compared to any other point. This measure would penalize unequal waiting times highly: the time Δt_i is often 0 when it is measured between the arrival to and departure from a timing point.

In this paper, we further consider the $LCSS_{\mathrm{thr}}$ and L^p distances.

4.4 Relationship Between L^p and $LCSS_{\mathrm{thr}}$

We first consider the case where the data are normalized: the value for each dimension is between 0 and 1. The L^p distance 1 between the trajectories that bound the dataset is 1, and the distance between identical trajectories is 0. When $LCSS_{\mathrm{thr}} = 0$ this means that for each dimension i, $|t_i - t'_i| \leq thr$. A distance of 1 implies that for each i, $|t_i - t'_i| > thr$. However, the L^p distances between trajectories that have $LCSS_{\mathrm{thr}}$ distances 1 and 0, correspondingly, can differ within an arbitrarily small value. Given $L^p = 0$ implies $LCSS_{\mathrm{thr}} = 0$. It is true that:

$$LCSS_{\mathrm{thr}} \cdot thr^p \leq L^p \leq (1 - LCSS_{\mathrm{thr}}) thr^p + LCSS_{\mathrm{thr}} \ .$$

For non-normalized data, the L^p distance is divided by $(max - min)^p$, and the threshold thr is divided by $(max - min)$, where min and max are the boundaries of the data for each dimension.

The measures can be used in combination. We observe that: (1) small L^p and small $LCSS_{\mathrm{thr}}$ imply that for most dimensions, the trajectories are close to each other; (2) small L^p and large $LCSS_{\mathrm{thr}}$ indicate that for most dimensions, the threshold is only slightly exceeded; (3) large L^p and small $LCSS_{\mathrm{thr}}$ indicate the presence of outliers; and (4) large L^p and large $LCSS_{\mathrm{thr}}$ mean that for most dimensions, the trajectories are far apart.

4.5 Weighted L^p Distance

It may be expected that the most recent segment of a trajectory is more important when predicting the near future that are older segments. In real road networks, the traffic is distributed unevenly in different parts of a network. For example, while a vehicle is crossing the downtown area, it will be faced with heavy traffic. Once it enters a highway, its speed pattern changes. To accommodate this, we propose to use a weighted L^p distance:

$$WL^p(f_{\mathrm{tr}}, f'_{\mathrm{tr}}) = \frac{1}{k} \sum_{i=1}^{cur} w_i (\Delta t_i - \Delta t'_i)^p \ ,$$

where $\overline{W} = \langle w_0, ..., w_{cur} \rangle$, $\Delta t_i \in f_{\mathrm{tr}}$, $\Delta t'_i \in f'_{\mathrm{tr}}$, and k is the number of non-zero weights w_i.

Weights may be constructed in several ways:

1. No weights, meaning that all parts are equally important, and $w_i = 1$ for all i.
2. Equal weights considering only the most recent sub-trajectory, where $w_0 ... w_{k-1} = 0$ and $w_k ... w_n = 1$. This generalizes the above scheme.
3. Linear weights, where $w_{i+1} = w_i + \alpha$, if $i \geq k-1$; $w_i = 0$, otherwise. This approach is a further generalization.
4. Polynomial weights of the form $w_{i+1} = \alpha w_i$ if $i \geq k$; $w_k = 1$; and $w_i = 0$, otherwise.
5. Pre-defined weights learned from historical data. Such correlation-based weights are discussed next.

We propose to use the Kendall τ rank correlation coefficient [16] for calculating correlation-based weights. This coefficient is applicable when non-linear correlations exist between the variables. The travel times on different segments are likely to have complex correlations. The variation τ_c makes adjustments for ties (equal values). It is defined as the number of pairs $(\Delta t_i, \Delta t_j)$ ordered concordant (n_c) minus the number of pairs ordered discordant (n_d) divided by the total number of pairs: $\tau_c = (n_c - n_d)/(n_c + n_d)$. A correlation table is built where each entry τ_{ij} is a correlation coefficient between points i and j. The table has $n(n-1)/2$ entries ($\tau_{ij} = \tau_{ji}$ and $\tau_{ii} = 1$). Assuming that the real-time trajectory f_{tr} ends at point p_{cur}, the query length is l, and j points ahead should be predicted. Then the weight w_i in the vector $\overline{W} = \langle w_{cur-l}, ..., w_{cur} \rangle$ is defined as follows:

$$w_i = \sum_{k=cur+1}^{cur+j} |\tau_{ik}| \overline{PV}_k \ .$$

Other correlation coefficients are possible (e.g., Pearson's). We use the Kendall rank correlation in our empirical evaluation, as higher correlations exist (Section 6).

4.6 Using f_{nn} for Prediction

The algorithm presented in Algorithm 1 explains how the f_{nn} can be used to predict future travel times. In the initialization phase, an initial prediction is formed (line 1). When a new position p_i at time t_i is received, the value $t_i \in f_{\mathrm{tr}}$ is calculated, and the prediction is revised. If the distance between the actual trajectory and the prediction is less than the known (conservative) minimum distance $d_{\min}$ plus a pre-defined threshold thr_{d}, the previous prediction f_{nn} is returned (lines 5–7). The threshold thr_{d} renders it possible to reduce the possibly time-consuming similarity searching. If needed, a new (approximate) f_{nn} is located in the trajectory database F_{tr} and is returned as the current prediction (line 8). If more than one f_{nn} exists, one is chosen at random.

Algorithm 1: *Predict*: continuous prediction

Input: Candidate set F_{tr}, threshold thr_{d}, distance function d.
Output: Prediction f_{nn}
1: $f_{\mathrm{nn}} \leftarrow$ initial prediction; $f_{\mathrm{tr}} \leftarrow \langle p_0, 0 \rangle$
2: **while** Vehicle is on the route **do**
3: Receive new p_i at t_i; $\Delta t_i \leftarrow t_i - t_{i-1}$
4: $f_{\mathrm{tr}} \leftarrow f_{\mathrm{tr}} \circ \langle p_i, \Delta t_i \rangle$ // new point
5: **if** $d(f_{\mathrm{tr}}, f_{\mathrm{nn}}) \leq d_{\min} + thr_{\mathrm{d}}$ **then**
6: **return** f_{nn}
7: **end if**
8: **return** $\operatorname{argmin}_{f'_{\mathrm{tr}} \in F_{\mathrm{tr}}} \max(d(f_{\mathrm{tr}}, f'_{\mathrm{tr}}), d_{\min} + thr_{\mathrm{d}})$
9: **end while**

5. SIMILARITY SEARCH

When there is no index present, the NNT method requires a sequential scan of the historical data every time the prediction needs to be revised: the distance is computed for each historical trajectory, and the one that minimizes this distance is returned. This procedure can be time-consuming, especially when the queries are long and the trajectory database is extensive. We propose a main-memory index that reduces the amount of the trajectories that have to be compared. The list-based index with the threshold algorithm (TA) [10] is extended to support continuous queries. We present an incremental algorithm ITA that updates the f_{nn} when a new point in f_{tr} is received. The index supports various distance functions, including $LCSS_{\mathrm{thr}}$, L^p, and WL^p, and it also allows NNT search using more than one distance function simultaneously.

5.1 Queries

A static query $Q_{\mathrm{SNN}} = \langle f_{\mathrm{tr}}[p_i, p_{\mathrm{j}}], d, \overline{PV}, thr_{\mathrm{d}} \rangle$ consists of a trajectory segment f_{tr} with times $(\Delta t_i, ..., \Delta t_{\mathrm{j}})$, a distance function d, a priority vector $\overline{PV}$, and a threshold thr_{d}. The most similar trajectory f_{nn} that minimizes the distance function d has to be retrieved with an approximation threshold thr_{d}.

A continuous query $Q_{\mathrm{CNN}} = \langle f_{\mathrm{tr}}[p_0, p_{cur-1}], \langle p_{cur}, \Delta t_{cur} \rangle, d, \overline{PV}, thr_{\mathrm{d}} \rangle$ receives subsequent points as the vehicle proceeds along its route. The initial query is a static query $\langle f_{\mathrm{tr}}[p_0, p_{\mathrm{l}}], d, \overline{PV}, thr_{\mathrm{d}} \rangle$, where $l \geq 1$ is the query length. Given the most recent query trajectory $f_{\mathrm{tr}}[p_0, p_{\mathrm{cur}-1}]$, an update of the query Q_{CNN} at time t_{cur} is the travel time Δt_{cur} between points $p_{\mathrm{cur}-1}$ and p_{cur}.

5.2 Data Structure

For a route $R = \langle p_0, ..., p_n \rangle$, a structure $\mathcal{L} = \langle L_1, ..., L_n \rangle$ with n lists, where each list L_i corresponds to the segment between timing points p_{i-1} and p_i, is created. Each entry j in a list L_i is a node $L_i[j] = \langle \Delta t_i, id \rangle$, where $\Delta t_i \in f_{\mathrm{tr}}$ is the ith travel time in the trajectory f_{tr}, and id is the identifier of the trajectory f_{tr}. List L_i is ordered by the travel time Δt_i of all trajectories in the database. Each entry in the list can be accessed randomly by the index j. The main memory has to store both the list structure and the array of the trajectories (it is not necessary to store the values Δt_i repeatedly—they can be accessed in constant time from the indexed array of trajectories by id and i). The trajectory that corresponds to an entry j in list L_i is denoted as $L_i[j]\uparrow$.

5.3 Search Algorithms

5.3.1 Threshold Algorithm

The Threshold Algorithm (TA), proposed in [10], uses the index structure $\mathcal{L}$ from above. It was originally designed to answer top-k queries, where the attributes of the objects in the database are stored in lists, one list per attribute, ordered by the ranks of the attributes. A monotonic, continuous function $g(v_1, ..., v_m)$ grades an object, where v_i is the value of an attribute from the corresponding list L_i. The lists can be searched using an arbitrary set of attributes, given the function g. The query trajectory is $\langle 0, ..., 0 \rangle$, the lists are accessed starting from the top in parallel, and the function g is evaluated for each object $L_i[j]\uparrow$ that has not been seen yet. A lower bound $B = f(\overline{v}_1, ..., \overline{v}_m)$ is calculated when a new entry is accessed, where $\overline{v}_i$ is the most recently seen value in the list L_i. When k objects $L_i[j]\uparrow$ are found such that $g(L_i[j]\uparrow) \leq B$, the algorithm stops. It is proved that the correct top-k objects are returned. The TA algorithm can be used to answer a static query Q_{SNN}, where $g(f'_{\mathrm{tr}}[p_i, p_j]) = d(f_{\mathrm{tr}}[p_i, p_j], f'_{\mathrm{tr}}[p_i, p_j])$, and $f_{\mathrm{tr}}[p_i, p_j]$ is the query.

We extend the TA algorithm to be incremental (ITA), and to an-

swer NNT queries from a given k-dimensional point. The focus is on top-1 queries, though the algorithm is easily extended to support top-k queries: rather than stopping when one NNT not further than the bound is found, the algorithm continues until k NNTs are found. The middleware that passes the incremental query to the index can be responsible for maintaining the history.

5.3.2 Static Query

The Q_{SNN} query processing algorithm is listed in Algorithm 2. The algorithm is similar to TA, except that only one list is searched. This is done so that binary search needs to be performed only once; furthermore, by choosing the optimal list L_s for search, the bound B is expected to grow faster than in TA for some measures d. For example, if $p > 1$ in the L^p distance, a greater difference at one point to the power of p grows faster than the sum of the differences to the power of p spread out among l points. The WL^p search is presumably more efficient when the list with the highest corresponding weight w_i is chosen.

Algorithm 2: *SortedLists.StaticSearch*: search once.

Input: Real-time trajectory $f_{\mathrm{tr}} = (t_i, ..., t_j)$, threshold thr_{d}, query length l.
Output: Prediction f_{nn}

1: $L_s \leftarrow$ search list in $\mathcal{L}$, $s \in [i, ..., j]$
2: $i \leftarrow BinarySearch(L_s, \Delta t_s)$
3: $f_{\mathrm{nn}} \leftarrow L_s[i]\uparrow$
4: **while** $d_l(f_{\mathrm{tr}}, f_{\mathrm{nn}}) - thr_{\mathrm{d}} > d(\Delta t_s, L_s[i])$ and L_s has more elements **do**
5: **if** $d_l(L_s[i]\uparrow, f_{\mathrm{tr}}) < d_l(f_{\mathrm{nn}}, f_{\mathrm{tr}})$ **then**
6: $f_{\mathrm{nn}} \leftarrow L_s[i]\uparrow$
7: **end if**
8: $i \leftarrow next(L_s, \Delta t_{cur})$
9: **end while**
10: **return** f_{nn}

5.3.3 Continuous Query

The incremental algorithm for processing the Q_{CNN} query is listed in Algorithm 3. It remembers the search results from the previous run. Denote f_{tr_k} as the query using the k most recent points. Then f_{nn_k} is the nearest neighbor of f_{tr_k}, d_k is the distance $d(f_{\mathrm{nn}_k}, f_{\mathrm{tr}_k})$, and B_k is the minimum bound for d_k. The trajectories f_{nn_k}, $k = 1, .., l$ are simultaneously searched using the 1 to l most recent points. The algorithm is applicable for all functions d that satisfy:

$$d_k(f_{\mathrm{tr}_k}, f'_{\mathrm{tr}_k}) = d_{k-1}(f_{\mathrm{tr}_{k-1}}, f'_{\mathrm{tr}_{k-1}}) + d_1(f_{\mathrm{tr}_1}, f'_{\mathrm{tr}_1}) \ . \quad (1)$$

The WL^p satisfies (1) in-between the iterations, while in the same iteration, the distance function d_k uses the weight vector $W_k = \langle w_{cur-l}, ..., w_{cur-l+k} \rangle$ that is different for each k.

The minimal bound B_k is maintained for each d_k during the search. When the query is issued for the first time, $B_k = 0$ for all $k = 1, ..., l$, and the f_{nn_k} are unknown. Otherwise, the algorithm starts with $B_k = B_{k-1}$, and $f_{\mathrm{nn}_k} = f_{\mathrm{nn}_{k-1}}$ or unknown, $k = 2, ..., l$. The starting point $L_{cur}[i]$ that is closest to Δt_{cur} is located in the search list L_{cur} using binary search (line 3). The algorithm continues until f_{nn_l} is located (lines 4–11), in each iteration shifting $L_{cur}[i]$ to be the next closest point to Δt_{cur} (line 10). Once $d_k \leq B_k$, the NNT $f_{\mathrm{nn}_k} = L_{cur}[i] \uparrow$ is located, and the bound is set to $B_k = d_k$ for the next iteration (line 15). If it is of interest to locate f_{nn_k} for all k, the algorithm can only stop when they all are found. If threshold $thr_{\mathrm{d}} > 0$, this is added to the bound B_l. At the end of an iteration, the bounds are calculated for the next iteration (lines 13–19).

Algorithm 3: *SortedLists.IncrementalSearch* (ITA): a new point is received in the current trajectory.

Input: Real-time trajectory f_{tr}, the most recent prediction f_{nn_l}, a new point $\langle p_{cur}, \Delta t_{cur} \rangle$, threshold thr_{d}, distance function d, query length l.
Output: Prediction f_{nn_l}

1: $f_{\mathrm{tr}} \leftarrow f_{\mathrm{tr}} \circ \langle p_{cur}, \Delta t_{cur} \rangle$
2: **if** $d_l(f_{\mathrm{tr}}, f_{\mathrm{nn}_l}) > B_{l-1} + thr_{\mathrm{d}}$ **then**
3: $i \leftarrow BinarySearch(L_{cur}, \Delta t_{cur})$; $B_1 \leftarrow d_1(\Delta t_{cur}, L_{cur}[i])$
4: **while** $d_l(f_{\mathrm{tr}}, f_{\mathrm{nn}_l}) - thr_{\mathrm{d}} > B_{l-1} + d_1(\Delta t_{cur}, L_{cur}[i])$ and L_{cur} has more elements **do**
5: **for all** $k \in 2..l$ **do**
6: **if** $d_k(L_{cur}[i]\uparrow, f_{\mathrm{tr}}) < d_i(f_{\mathrm{nn}_k}, f_{\mathrm{tr}})$ **then**
7: $f_{\mathrm{nn}_k} \leftarrow L_{cur}[i]\uparrow$
8: **end if**
9: **end for**
10: $i \leftarrow next(L_{cur}, \Delta t_{cur})$
11: **end while**
12: **end if**
13: **for all** $k \in 2..l$ **do**
14: **if** f_{nn_k} is found **then**
15: $B_k \leftarrow d_{k-1}(f_{\mathrm{tr}}, f_{\mathrm{nn}_{k-1}})$
16: **else**
17: $B_k \leftarrow B_{k-1} + d(\Delta t_{cur}, L_{cur}[i])$
18: **end if**
19: **end for**
20: **return** f_{nn_l}

The lists are visualized in Figure 1. The query length $l = 4$, and $\Delta t''_{cur} \in f_{\mathrm{nn}_4}$ is used to predict Δt_{cur}. When the actual Δt_{cur} becomes known, the prediction is revised. The value $\Delta t'_{cur} \in f_{\mathrm{nn}_3}$ limits the search: the new f_{nn_4} is listed no further than $d_s = |\Delta t_{cur} - \Delta t'_{cur}|$ in the list L_{cur}. With $LCSS_{\mathrm{thr}}$ it is never necessary to search more than within $\min(thr, d_s)$ from the query point.

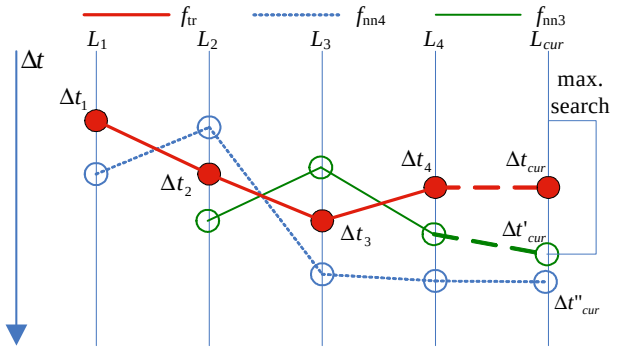


Figure 1: List-based indexing

The correctness of the algorithm, when all lists are searched in parallel and the query is static, is proved elsewhere [10]. Though our proposed both static and incremental algorithms search only one list, the principles are the same: the lower bound is calculated according to the most recently read values in the list that is searched, and the other points do not contribute to the increase of the bound. Next we prove that the incremental algorithm returns the correct result.

LEMMA 1. *Given a query of length l on points $\langle p_{cur-l+1}, ..., p_{cur} \rangle$, $f_{\mathrm{tr}} = \Delta t_{cur-l+1}, ..., \Delta t_{cur}$, and a database that contains the corresponding l ordered lists $L_{cur-l+1}$,...,L_{cur} (as described above). Then algorithm ITA returns the f_{nn_l} that minimizes the*

distance $d_l = d(f_{\mathrm{tr}}, f_{\mathrm{nn}l})$ (with an allowed approximation threshold thr_{d}).

PROOF. The case when all entries in the list L_{cur} are accessed is trivial: the $f_{\mathrm{nn}k}$ that are within the minimum distance from the query are found for each k. Otherwise, the algorithm has found an $f_{\mathrm{nn}k}$ when $d_k(f_{\mathrm{tr}}, f_{\mathrm{nn}k}) \leq B_k$, if $thr_{\mathrm{d}} = 0$, or when $d_k(f_{\mathrm{tr}}, f_{\mathrm{nn}k}) \leq B_k + thr_{\mathrm{d}}$, if $thr_{\mathrm{d}} > 0$. We prove it when $B_k \geq 0$ using the induction principle, where the iterator m is the sequence number of the most recently received point $\langle p_m, \Delta t_m \rangle$ in the query. First let's consider the case when $thr_{\mathrm{d}} = 0$. At step 0, $B_k = 0$ for all k, and the algorithm works as in [10], except that ITA searches only one list, and the lower bound B_k is $d(0, ..., 0, L_l[i])$. Assume that the algorithm works correctly at step $m-1$: after it has stopped, if $d_k(f_{\mathrm{tr}}, f_{\mathrm{nn}k-1}) \leq B_{k-1}$, the trajectory $f_{\mathrm{nn}k-1}$ is an NNT of the query of length $k-1$; otherwise, $f_{\mathrm{nn}k-1}$ is still not located, and the actual shortest distance $d_{k-1} \geq B_{k-1}$. Now consider step m, renaming B_k from the previous iteration to B'_k. Using (1), for any trajectory f'_{tr}:

$$d_k(f_{\mathrm{tr}}, f'_{\mathrm{tr}}) = d_{k-1}(f_{\mathrm{tr}}, f'_{\mathrm{tr}}[p_{m-k+1}, p_{m-1}]) +$$
$$d(f_{\mathrm{tr}}[p_m], f'_{\mathrm{tr}}[p_m]) \geq B'_{k-1} + d(f_{\mathrm{tr}}[p_m], f'_{\mathrm{tr}}[p_m]) = B_k \ .$$

This means that going further in the list L_m (increasing $d(f_{\mathrm{tr}}[p_m], f'_{\mathrm{tr}}[p_m])$), the current minimum bound B_k will only increase. Once the current $f_{\mathrm{nn}k}$ is within a smaller distance from the query than the bound, $d_k(f_{\mathrm{tr}}, f'_{\mathrm{tr}}) \leq B_k$, no closer trajectories exist. If the algorithm terminates before $f_{\mathrm{nn}k}$ is located, for the next step, the bound B_k is set to be $B_k = B'_{k-1} + d(f_{\mathrm{tr}}[p_m], f'_{\mathrm{tr}}[p_m])$, which is the currently known smallest possible distance. The trajectory $f_{\mathrm{nn}l}$ is always found as this is the stopping condition of the iteration.

Approximate NNT search is also discussed elsewhere [10]. If the algorithm stops when $d_l(f_{\mathrm{tr}}, f_{\mathrm{nn}l}) \leq B_l + thr_{\mathrm{d}}$, the closest trajectory that can exist is within distance B_l, i.e., no further away than thr_{d} from the returned approximate $f_{\mathrm{nn}l}$. □

Figure 2 illustrates ITA by an example. The identifiers of the trajectories are f_i, $i = 1, ..., 4$, and each arrow points to a value Δt_i in the corresponding list L_i. Assume that a query $\langle f_{\mathrm{tr}} = \langle ...[p_{k-3}, 3], [p_{k-2}, 4], [p_{k-1}, 3] \rangle, [p_k, 3], d = L^1, thr_d = 0 \rangle$ is issued. The query length is $l = 3$, and the most recent NNT is f_3, where $d(f_3, f_{\mathrm{tr}}[p_{k-3}, p_{k-1}]) = 6$. The initial bounds before list L_k is first accessed are $B_1 = 0$, $B_2 = 4$, $B_3 = 5$ (they were obtained in the previous steps, when searching lists L_{k-1} and L_{k-2}). After the binary search in L_k, where the closest entry $f_3 \rightarrow 4$ is located, the bounds are updated to be $B_1 = 1$ $B_2 = 5$, $B_3 = 6$, and $d(f_3, f_{\mathrm{tr}}[p_{k-3}, p_{k-1}]) = 6$. As this distance equals the value of B_3, the search stops, and the current nearest neighbor is still f_3.

$\mathbf{L_{k-3}}$	$\mathbf{L_{k-2}}$	$\mathbf{L_{k-1}}$	$\mathbf{L_k}$
f_1->1	f_2->3	f_1->1	f_4->1
f_3->2	f_1->4	f_2->2	f_1->1
f_4->5	f_3->5	f_3->5	f_3->4
f_2->7	f_4->6	f_4->6	f_2->6

Figure 2: ITA search example

5.4 Memory Resources

We expect that the main memory is sufficient to store the index and data for transport-related applications. The travel times of the historical trajectories are stored in an indexed array of size $n \times m$, where m is the number of trajectories and n is the total number of points in one trajectory. Allowing 2 bytes for Δt_i, with a database containing 5000 trajectories on a route of length 50, the data takes up only 500 kb. The index as a minimum stores the identifiers of the trajectories in $n \times m$ positions. If each identifier is allowed 2 bytes, this results in $4nm\,\mathrm{kb} = 1\,\mathrm{MB}$. The algorithm itself needs to remember only a small constant number of values for each real-time trajectory. Hence, even with few hundreds of different routes, the data does not need to be looked up on the disk during the search.

The index can also be used by different applications than transport systems, where very large databases have to be handled. In general, the prediction model can be applied to any time series where one dimension is fixed (e.g., the times when a measurement is taken), and where the past behavior is expected to predict the future. A disk-based TA is available elsewhere [10]. A disk-based ITA is also possible; however, the discussion is beyond the scope of this paper.

6. EMPIRICAL EVALUATION

The NNT-based prediction was empirically evaluated using both real and synthetic data, and the prediction accuracy was measured. Furthermore, CPU time was measured for ITA, the proposed search algorithm, comparing it to both TA and sequential scan (SS).

6.1 Experimental Setup

Our data generator produces trajectories on a route of length n, clustered into $cnum$ overlapping clusters. Each point on the center of the cluster C_i, $i = 0, ..., cnum - 1$, is uniformly distributed in the interval $[start, end]$, where the start of the interval is deterministic, $start = min + (max - min)i/cnum$, and the length of the interval is a random multiple of $(max - start)/c$. Here $[min, max]$ is the range of values that Δt_i can take. Each cluster has $m/cnum$ trajectories, where m is the total number of trajectories. Each point in a trajectory is randomly displaced from the center of the cluster within the maximum radius r. A cluster is further divided into two overlapping clusters, each of them with a radius of $3/4r$. A given percentage of point outliers, p_{out}, uniformly distributed in $[min, max]$, are picked randomly among all points. A given percentage of cuts in the trajectories, p_{cut}, introduces random "jumps" to another cluster at random points. A change of clusters at pre-defined points for all trajectories is defined by the number seg. If $seg > 1$, the trajectory is divided into equal segments, every second segment being assigned to another randomly chosen cluster. More details and the source code can be found at `http://transdb.cs.aau.dk/` under "publications". The default parameters are listed in Table 1.

The real data was collected from the buses traveling on line 1 in Aalborg, Denmark. A sub-route (having the most data) with 23 stops and 228 traversals was chosen, and a *timing point* is either an arrival to or a departure from a bus stop. The mean travel or waiting time per segment $[p_i, p_{i+1}]$ is 43.5 s, and the average standard deviation of Δt_i is 27.6 s.

6.2 Evaluation of the Similarity Measures

A set of experiments was performed, where the value of one parameter is varied in each experiment, and the average prediction error err per point is measured. The error with two different $\overline{PV}$ is evaluated: predicting the arrival time only for the nearest point in the future ("next", $\overline{PV} = \langle 1, 0, ..., 0 \rangle$), and predicting the full future trajectory ("all", $\overline{PV} = \langle 1, ..., 1 \rangle$). The distances L^1 ("L") and $LCSS_{\mathrm{thr}}$ ("$LCSS$") were tested in most experiments. Furthermore, we experiment with incremental weights $W = \langle 1, ..., n \rangle$ ("WL"), correlation-based weights ("CWL"), and different thresholds thr. The prediction is revised every time a new point is received. The parameter p is set to 1 as the initial tuning showed

Table 1: Default parameters

Route length n	50
Query length l	5
Number of trajectories m	500
Point and trajectory distortion p_{out}, p_{cut}	5%, 50 %
Number of clusters $cnum$	10
Time interval per segment $[min, max]$	$[60, 600]$ s
Cluster radius r	20 s
Threshold thr in $LCSS_{\text{thr}}$	10 s
Prediction threshold thr_{d}	0
Weight increment $inc = w_{i+1} - w_i$	0
Number of segments of f_{tr} in different clusters seg	1
Prediction error err	L^1

that increasing p reduces the prediction accuracy. The combination of the WL^p and $LCSS_{\text{thr}}$ measures does not appear to improve the prediction accuracy. This method requires a carefully designed switching mechanism, which is beyond the scope of this paper.

6.2.1 Real Data

The query length l was varied from 1 to 25 points in steps of 2, and the performance of various measures d was evaluated (Figures 3(a), 3(b)). The prediction error is quite similar for all distance measures, and it decreases when the query length grows. The correlation-based weights yield the best prediction, when used together with long queries. Such weights are the most flexible and allow to select the f_{nn} based on those points that are expected be the most informative. The average optimal distance to f_{onn} (the minimal err) was measured to be around 10 s—only with very long queries does the actual error approach this number. In the subsequent experiments we use synthetic data, which allows to test various parameters and search in a more extensive dataset.

6.2.2 Varying Threshold

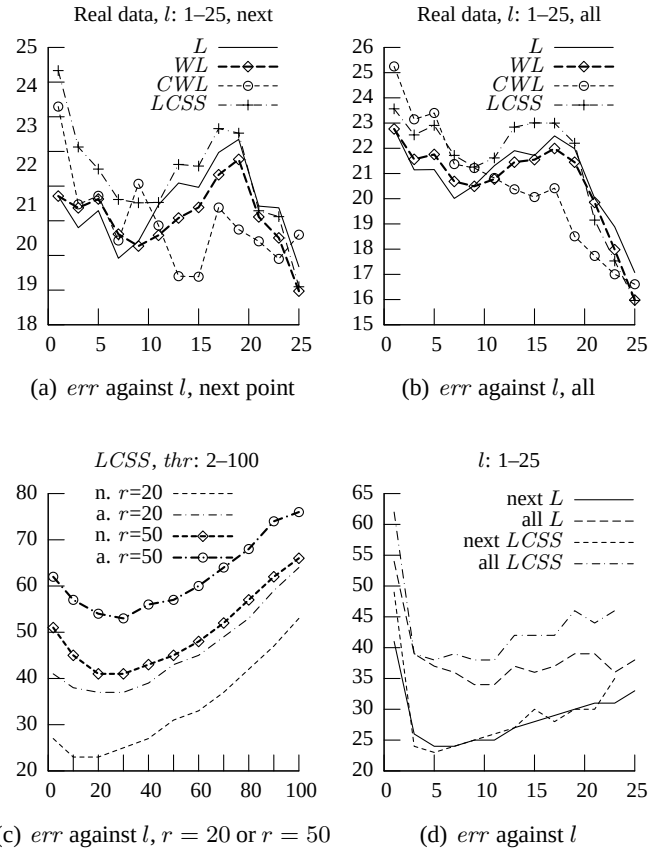
The threshold thr in $LCSS_{\text{thr}}$ distance was varied from 10 to 100 s in steps of 10 s (Figure 3(c)). The optimal threshold for the data generated with default parameters is around 10 s, though it increases when the within-cluster deviation r increases. The threshold is directly related to the variance of "white noise", i.e., the unpredictable deviation from the travel pattern.

6.2.3 Varying Query Length

The length of the query was varied from 1 to 25 points in steps of 2 (Figure 3(d)). For both L^p and $LCSS_{\text{thr}}$ distances, the short-term prediction is the most accurate when the query length is around 5 points, while for longer term prediction, longer queries (10 points) are preferred. Both distance measures give similar prediction error for short-term prediction, and the L^p distance is slightly better for long term prediction. The reason for the decrease of accuracy when too long queries are used is that the trajectories may change their patterns unexpectedly—long queries do not adjust to rapidly changing conditions, while shorter queries are more flexible.

6.2.4 Testing Weight Assignment Methods

We have evaluated the WL^p measure using correlation-based weights ("CWL"), incremental weights ("WL"), and no weights ("L"), when $seg = 5$: each trajectory changes its cluster every 10 points, though some of the clusters periodically repeat. The query length l was varied from 1 to 25 points in steps of 2 (Figure 4(a)). Similarly to the real data, the correlation-based weights give the



(a) err against l, next point (b) err against l, all

(c) err against l, $r = 20$ or $r = 50$ (d) err against l

Figure 3: Prediction error

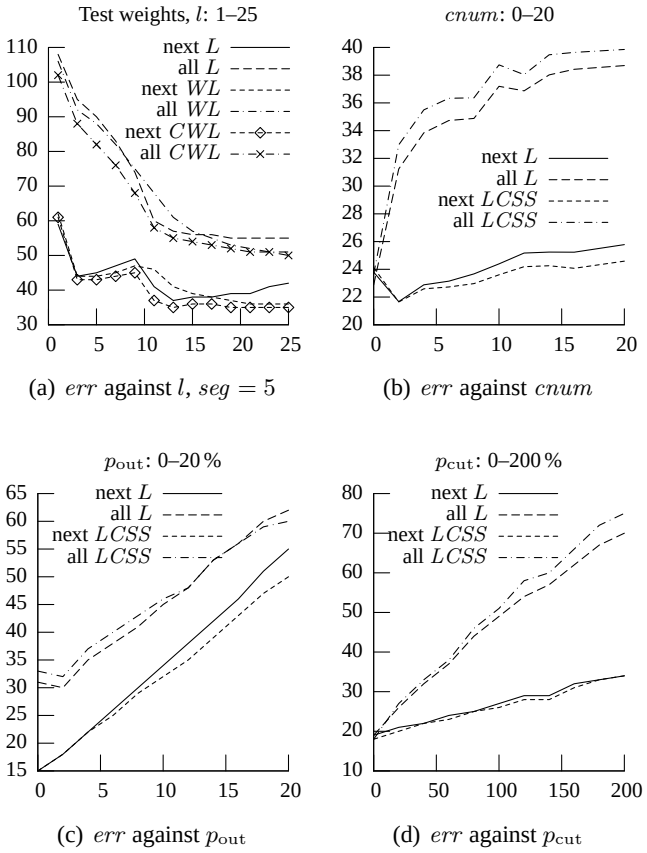
best results, when long queries are allowed. The WL^p measure can achieve similar accuracy, and L^p is the least accurate. This is because only some segments of trajectories are correlated, while the others should be left out. Very short queries are not sufficient, as they do not have time to "learn" the repeating pattern. However, when no correlations are discovered, the correlation-based weights diverge to equal weights for all points. The observations lead to the general conclusion that the weights and the query length should be chosen according to the properties of the data.

6.2.5 Varying the Number of Clusters

The number of clusters $cnum$ was varied from 0 to 20 in steps of 2 (Figure 4(b)). For all measures, the prediction error grows together with the number of clusters—the overlap of the clusters increases, and the points are spread out in a larger space. The increase is especially rapid for the prediction of the full trajectory: the choice of the f_{nn} from an incorrect cluster has a significant effect on the prediction error.

6.2.6 Varying the Percentage of Outliers

The percentage of outliers p_{out} was varied from 0 % to 20 % in steps of 2 % (Figure 4(c)). The prediction error increases rapidly in all cases, as more points in the trajectories become unpredictable, and these points also prevent the indentification of similar trajectories that would give adequate predictions. We have noticed that increasing the query length does not improve the prediction either—presumably, more outliers in long queries lead to the selection of incorrect patterns (trajectories from further away clusters).



(a) err against l, $seg = 5$ (b) err against $cnum$

(c) err against p_{out} (d) err against p_{cut}

Figure 4: Prediction error

6.2.7 Varying the Percentage of Trajectory Cuts

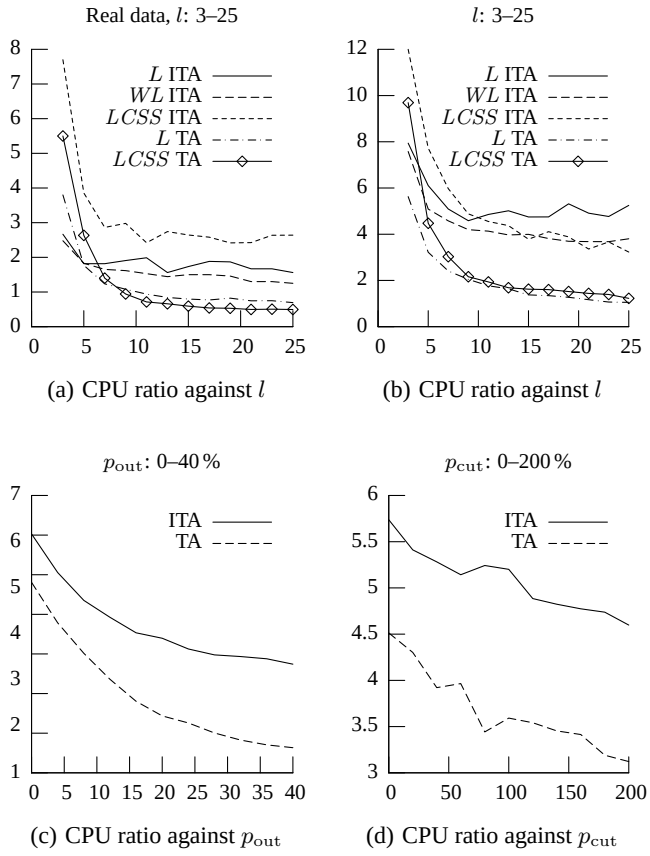
The number of random cuts p_{cut} in the trajectories was varied (at a random point a trajectory "jumps" from its current cluster to a new one) from 0 % to 200 % of m in steps of 20 % (Figure 4(d)). The prediction error increases with the number of cuts for both types of distance measures, especially when predicting further away into the future. This is as expected because the segment of the trajectory after the cluster has changed is unpredictable. The observations lead to the conclusion that only short-term prediction is useful when p_{cut} is high.

6.3 Evaluation of CPU Performance

The experiments were performed on a PC with an Intel Pentium 3 996 MHz processor, 512 kb of RAM, and the Windows XP OS. The code was written in C#, running on the .NET Framework 2.0. The performance measure is the ratio of the total CPU time for SS against TA and ITA, CPU(SS)/CPU(TA) and CPU(SS)/CPU(ITA). The CPU performance was evaluated for the different distance measures, using both real and artificial data, and varying the default parameters l, $cnum$, m, p_{out}, and p_{cut}, as in the experiments that evaluate the prediction accuracy. When $l = 1$, the list search uses $\log_2 m$ time instead of m as does SS. This result is omitted from the graphs.

The real data requires similar CPU time as the data generated with no clusters (Figures 5(a) and 6(a)): the improvement of ITA with L^p, when compared to SS, is nearly a factor of 2 in most cases, and the queries that look only into the most recent past are significantly more efficient. The incrementally updated bounds in ITA allow to stabilize the CPU time, even when the query length increases, while the efficiency of TA drops more noticeably. Using weights decreases performance slightly: they do not allow the distance function to be calculated incrementally in the same iteration. The $LCSS_{\text{thr}}$ measure is the most efficient with real data, since the threshold limits the search.

ITA with synthetic data, when queries are long, gives the improvement ratio of 4 to 6 (Figure 5(b)), while TA only works well with very short queries. Though the CPU time increases linearly together with the increasing number of clusters (Figure 6(a)), the increase is much slower when compared to SS. TA gains much less in performance with clustered data, as it has no prediction memory. With a high number of outliers (Figure 5(c)), the performance of TA approaches the performance of SS, and ITA stays efficient. When the future becomes less predictable (Figure 5(d)), the incremental search becomes less efficient, though the advantage of ITA versus TA remains significant. Only with large amounts of data and clusters is TA preferable over ITA (Figure 6(b)) due to a large number of candidate NNTs for each query (the number of clusters did not increase together with the total amount of trajectories). When queries are long, ITA always preserves the advantage.



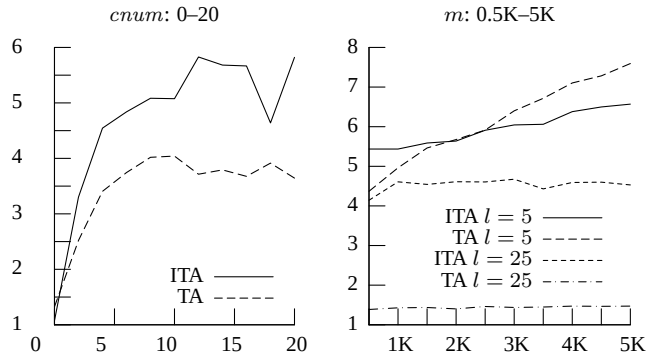
(a) CPU ratio against l (b) CPU ratio against l

(c) CPU ratio against p_{out} (d) CPU ratio against p_{cut}

Figure 5: CPU performance

7. CONCLUSION

This study reported upon in this paper aims to explore the suitability of similarity measures for history-based travel-time predictions for vehicles traveling on known routes. The underlying hy-



(a) CPU ratio against $cnum$ (b) CPU ratio against m

Figure 6: CPU performance

pothesis was that similar trajectories from the past can be used to predict the future travel behavior of a vehicle. Although proposals exist for similarity measures for trajectories of moving objects, the settings corresponding to collective transport applications have not yet been explored. We propose variations of the weighted L^p-norm, WL^p, and Longest Common Subsequence, $LCSS_{\mathrm{thr}}$, which are applicable for the comparison of trajectories that are restricted to pre-defined routes. Furthermore, we propose a main-memory index for the efficient access to historical trajectories.

Empirical studies were performed with both real and synthetic data. The trajectories in the real data do not form apparent clusters, and they appear to be quite unpredictable. As a result, both the prediction accuracy and performance are worse when compared to the synthetic data, where the trajectories can be grouped into clusters. Use of carefully chosen weights offers advantage over the L^p-norm, especially when only some segments of the trajectories are correlated. The $LCSS_{\mathrm{thr}}$ distance is in many cases the most efficient to compute; it is also slightly less accurate than the other measures considered. The proposed incremental algorithm, ITA, proves to be robust and it processes the query several times faster when compared to the sequential scan. When the data is "predictable" (partially similar trajectories exist), and this is what we expect when using similarity search, the incremental algorithm (i.e., ITA) is significantly more efficient than the static threshold algorithm (i.e., TA).

As a continuation of this work, we are developing an adaptive algorithm that evaluates the accuracy of a library of available prediction algorithms in real time and then uses the prediction algorithm that yields the most accurate predictions for the near past. The objective is to always use the most accurate prediction algorithm given the (ever changing) environment.

8. REFERENCES

[1] P. Bakalov, E. Keogh, and V. Tsotras. TS2-tree—an efficient similarity based organization for trajectory data. *Proc. of the Annual ACM International Symposium on Advances in Geographic Information Systems*, pp. 1–4, 2007.

[2] Y. Bin, Y. Zhongzhen, and Y. Baozhen. Bus arrival time prediction using support vector machines. *Journal of Int. Transp. Systems: Technology, Planning, and Operations*, 10(4): 151–158, 2006.

[3] B. Bollobas, G. Das, D. Gunopulos, and H. Mannila. Time-series similarity problems and well-separated geometric sets. In *Proc. of the Annual Symposium on Computational Geometry*, pp. 454–456, 1997.

[4] F. Cathey and D. Dailey. A prescription for transit arrival/departure prediction using automatic vehicle location data. *Transp. Res. Part C*, 11(3-4): 241–264, 2003.

[5] L. Chen and R. Ng. On the marriage of Lp-norms and edit distance. In *Proc. of the International Conference on Very Large Data Bases*, pp. 792–803, 2004.

[6] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp. 491–502, 2005.

[7] S. I.-J. Chien, Y. Ding, and C. Wei. Dynamic bus arrival time prediction with artificial neural networks. *Trans. Engrg.*, 128: 429–438, 2002.

[8] E.-H. Chung and A. Shalaby. Expected time of arrival model for school bus transit using real-time global positioning system-based automatic vehicle location data. *Journal of Int. Transp. Systems*, 11: 157–167, 2007.

[9] D. Dailey, S. Maclean, F. Cathey, and Z. Wall. Transit vehicle arrival prediction: An algorithm and a large scale implementation. *Transp. Res. Rec., Transportation Network Modeling*, pp. 46–51, 2001.

[10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Comp. and Syst. Sciences*, 66(4): 614–656, 2003.

[11] J. R. Hee and L. R. Rilett. Bus arrival time prediction using artificial neural network model. In *Proc. of the International IEEE Conference on Int. Transp. Systems*, pp. 988–993, 2004.

[12] J. Hwang, H. Kang, and K. Li. Spatio-temporal similarity analysis between trajectories on road networks. In *Proc. of the International Workshop on Conceptual Modeling for Geographic Information Systems*, pp. 280–289, 2005.

[13] J. Hwang, H. Kang, and K. Li. Searching for similar trajectories on road networks using spatio-temporal similarity. In *Proc. of the East-European Conference on Advances in Databases and Information Systems*, pp. 282–295, 2006.

[14] R. H. Jeong. *The Prediction of Bus Arrival Time Using Automatic Vehicle Location Systems Data*. Doctoral dissertation, Texas A&M University, 2004.

[15] R. Kalaputapu and M. J. Demetsky. Modeling schedule deviations of buses using automatic vehicle location data and artificial neural networks. *Transp. Res. Rec.*, 1497: 44–52, 1995.

[16] M.G. Kendall. Rank Correlation Methods. *Charles Griffin & Co. Ltd.*, 166 p, 1962.

[17] W.-H. Lin and J. Zeng. An experimental study on real time bus arrival time prediction with GPS data. *Journal of Transportation Research Board*, pages 101–109, 1999.

[18] T. Park, S. Lee, and Y.-J. Moon. Real time estimation of bus arrival time under mobile environment. In *Proc. of International Conference on Computational Science and its Applications*, pp. 1088–1096, 2004.

[19] N. Pelekis, I. Kopanakis, G. Marketos, I. Ntoutsi, G. Andrienko, and Y. Theodoridis. Similarity Search in Trajectory Databases. In *Proc. of the International Symposium on Temporal Representation and Reasoning*, pp. 129–140, 2007.

[20] B. Predic, D. Stojanovic, S. Djordjevic-Kajan, A. Milosavljevic, and D. Rancic. Prediction of bus motion and continuous query processing for traveler information services. In *Proc of the East European Conference on Advances in Databases and Information Systems*, pp. 234–249, 2007.

[21] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acoustics, Speech and Signal Processing*, 26(1):43Ű-49, 1978.

[22] A. Shalaby and A. Farhan. Prediction model of bus arrival and departure times using AVL and APC data. *Journal of Pub. Transp.*, 7: 41–61, 2004.

[23] D. Tiesyte and C. S. Jensen. Recovery of vehicle trajectories from tracking data for analysis purposes. In *Proc. of the European Congress and Exhibition on Intelligent Transport Systems and Services*, pp. 1–12, 2007.

[24] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proc. of the International Conference on Data Engineering*, pp. 673–684, 2002.