Abstract
We study the problem of finding the $k$ most similar trajectories in a graph to a given query trajectory. Our work was inspired by the work of Grossi et al. [6] that considered trajectories as walks in a graph in which each visited vertex is accompanied by a time-interval. Grossi et al. define a similarity function which is able to capture both temporal and spatial aspects. We modify this similarity function in order to derive a spatio-temporal distance function for which we can show that a specific type of triangle inequality is satisfied. This distance function builds the basis for our index structure which is able to quickly answer queries for the top-$k$ most similar trajectories. Our evaluation on real-world and synthetic data sets shows that our new approaches outperform the baselines with respect to indexing time, query time, and quality of results.

1 Introduction

Similarity of trajectories is attracting increasing attention in the scientific literature [1, 2, 11, 4, 3, 10, 12] since it is the basis for trajectory data mining. Often the trajectories of interest are related to a graph.

- **Tourism:** Persons share their side-seeing tours. The locations are points-of-interests (POI) and the intervals are the duration person stays at a POI. These travel logs are shared in social communities. A query is a request for a recommendation.

- **Animal behavior:** Consider wild live that is tracked using GPS. The living space of the animal is divided into zones. The goal is to identify similarities in animal behaviors. Vertices represent either specific locations like waterhole or feeding place, or territories of animals.

- **Website browsing analyses:** Web users following links on a website. The goal is to find similar browsing behavior.

- **Traffic and crowd analysis:** Here the goal is to identify person or vehicle flows at specific times through predefined areas. Vertices represent these areas.

These applications have in common that one is interested in finding a set of the most similar trajectories to a given one. An important basis for trajectory mining like clustering, outlier detection, classification or prediction tasks is the selection of an adequate similarity measure or distance function, respectively. Here, we are interested in similarity which takes spatial and temporal aspects into account. Two trajectories are close to each other if they visit vertices nearby to each other at about the same time.

Our work has been inspired by Grossi et al. [4], who claim to provide the first similarity measure for two trajectories in a graph of different length taking into account spatial and temporal similarity, and which can be computed in linear time with respect to the length of the trajectories. The authors defined a trajectory to be a walk in the graph for which each visited vertex is associated with a time interval.

Based on the trajectory notion in [6], we introduce a new spatio-temporal similarity. The corresponding distance function provides the basis for efficient algorithms answering top-$k$ similarity queries. A top-$k$ trajectory query $(Q, s)$ consists of a trajectory $Q$ and a time interval $s$. The result contains all trajectories in $T$ having one of the $k$ highest similarities to $Q$ with respect to the time interval $s$. Our contributions are the following:

1. We introduce a spatio-temporal similarity function and show that for the corresponding distance function the triangle-inequality holds under certain conditions. The similarity computation for two trajectories only needs linear time with respect to the length of the longest trajectory.

2. We use the advantageous properties of our new similarity function to design indices that can be constructed very efficiently and use linear memory with respect to the number of trajectories. The indices are based on spatial as well on temporal aspects.
filtering and allow heuristic top-k similarity queries with low running times and high quality of the results. Additionally, we apply upper bounding, which allows a direct, highly efficient query even without the need for a preprocessed index data structure. In the latter case the output is exact.

3. We evaluate our new approaches on real-world and synthetic data sets. Our new approaches outperform the baselines (including [6]) with respect to indexing time by several orders of magnitude. In most cases the query time and the quality of the results are better or on-par to the baseline algorithms.

1.1 Related Work Since trajectory similarity is of high interest for many data analytics tasks, many different similarity measures have been used, e.g., based on dynamic time warping, euclidean distances or edit distances. For trajectory analysis in networks, many approaches have concentrated on the spatial similarity only.

Hwang et al. [7] have suggested a similarity measure which is based on the network distance measuring spatial and temporal similarity. However, a set of nodes need to be selected in advance, and spatial similarity then means passing through the same nodes at the same time. Xia et al. [13] use a similarity measure for network constrained trajectories based on an extension of the Jaccard similarity. As similarity measure they use the product of spatial and temporal similarity. Tiakas et al. [11] [12] suggest a weighted sum of spatial and temporal similarity. Their similarity function works for two trajectories with the same length, and can be computed in linear time with respect to the length of the given trajectories.

Our work has been inspired by Grossi et al. [8]. They suggest a spatio-temporal similarity measure for two trajectories in a graph of different length. If the pairwise distances are given, then the measure can be computed in linear time with respect to the length of the trajectories. The authors also suggest an algorithm for answering the top-k trajectory query problem. For speeding up the computations they suggest an indexing method based on interval trees and a method to approximate their similarity measure. A more detailed description of their work and a comparison to our approach is provided in section 5.

Another way to approach the problem is to use a distance measure based on the discrete Fréchet distance, or dynamic time warping, which optimize over all vertex-mappings between the two trajectories that respect the time-ordering, where the underlying metric would be derived from the shortest-path metric given by the graph. Near-neighbor data structures have been studied theoretically with specific conditions on the underlying graph and the length of the queries, see [5, 8].

2 Preliminaries

An undirected and weighted graph $G = (V, E, c)$ consists of a finite set of vertices $V$, a finite set $E \subseteq \{(u, v) \subseteq V \mid u \neq v\}$ of undirected edges and a cost function $c : E \rightarrow \mathbb{R}_{\geq 0}$ that assigns a positive cost to each edge $e \in E$. A walk in a graph $G$ is an alternating sequence of vertices and edges connecting consecutive vertices. For notational convenience we sometimes omit edges. The length of a walk $(v_1, e_1, v_2, \ldots, e_k, v_{k+1})$ is $k$, i.e., the number of edges traversed. A path is a walk that visits each vertex at most once. The cost of a walk or path is the sum of its edge costs. Let $d(u, v)$ denote the shortest path distance between $u, v \in V$.

**Definition 1. (Trajectory) Let $G = (V, E, c)$ be an undirected, weighted and connected graph. A trajectory $T$ is a sequence of pairs $((v_1, t_1), \ldots, (v_k, t_k))$, such that for $1 \leq i \leq k$ the pair $(v_i, t_i)$ consists of $v_i \in V$ and a discrete time interval $t_i = [a_i, b_i]$ with $a_i, b_i \in \mathbb{R}$, $a_i < b_i$ and $a_{i+1} = b_i$ for $1 \leq i < k$.**

The starting time of $T$ is $T.start = t_1$ and the ending time $T.end = t_k$. We denote with $I(T)$ the total interval in which trajectory $T = ((v_1, t_1), \ldots, (v_k, t_k))$ exists, i.e., from $T.start$ to $T.end$. For a trajectory $T$ and a time interval $t = [a, b]$ we define $T[t]$ as the time-restricted trajectory that is intersected with $t$, i.e., $T[t] = ((v_i, t'_i), \ldots, (v_j, t'_j))$ with $v_i$ (vj) being the first (last) vertex of $T$ such that for $t_i = [a_i, b_i]$ it holds that $b_i > a$ (and for $t_j = [a_j, b_j]$ $a_j < b$, resp.), $t'_i = \max\{t_i, a\}$ and $t'_j = \min\{t_j, b\}$. We assume for $T = ((v_1, t_1), \ldots, (v_k, t_k))$ that $v_i \neq v_{i+1}$ for all $1 \leq i < k$. We say trajectory $T$ intersects a time interval $t$ if there is a $(v_i, t_i) \in T$ with $t_i \cap t \neq \emptyset$.

3 Spatio-Temporal Similarity

In this section we define our new similarity function for trajectories on networks based on the work of [9]. The goal of the similarity is to capture both temporal and spatial aspects, such that if two trajectories are often in close spatial distance, i.e., visiting vertices that are close to each other, at the same time, then the similarity should be high.

**Definition 2. Let $T = ((v_1, t_1), \ldots, (v_k, t_k))$ and $Q = ((u_1, s_1), \ldots, (u_k, s_k))$ be two trajectories, and $s$ a time interval. We define the similarity of $Q$ and $T$ in the
time interval $t$ as

$$\text{Sim}(Q, T, s) = \frac{1}{|s|} \cdot \sum_{(v_i, t_i) \in T} |s \cap t_i \cap s_j| \cdot e^{-d(v_i, u_j)}.$$  

Notice that for two trajectories $T$ and $Q$, and a time interval $s$, it holds that $0 \leq \text{Sim}(Q, T, s) \leq 1$. $\text{Sim}(Q, T, s)$ is minimal if the intersection of the intervals is empty for all intervals. In this case $\text{Sim}(Q, T, s) = 0$.

**Lemma 3.1.** Let $Q$ and $T$ be trajectories and $s$ a time interval. It holds that

1. $\text{Sim}(Q, T, s) = \text{Sim}(T, Q, s)$, and
2. let $s \subseteq \mathcal{I}(Q)$, then $Q[s] = T[s]$ if and only if $\text{Sim}(Q, T, s) = 1$.

**Proof.** (1.) The shortest path metric is symmetric, i.e., $d(u, v) = d(v, u)$ for all $u, v \in V$. The summation is over the same pairs of $(v_i, t_i) \in T$ and $(u_j, s_j) \in Q$, and the intersection of the intervals is commutative. Therefore, the result follows.

2. $\Rightarrow$: Notice that if $Q[s] = T[s]$ in each step of the summation $e^{-d(u, v)} = 1$. Because $s \subseteq \mathcal{I}(Q)$, the result of the summation is $s$ and normalization is $1$.

$\Leftarrow$: Assume that $\text{Sim}(Q, T, s) = 1$ but $Q[s] \neq T[s]$, i.e., $Q[s]$ and $T[s]$ differ in the vertices they visit or the times when they visit them. In the first case, due to the strictly positive edge weights, there is a vertex pair such that $e^{-d(u', v')} < 1$, however for all other vertex pairs the value $e^{-d(w', v')}$ is at most $1$. Because the intervals are intersected with the interval $s$ the total sum will be less than $|s|$ and leads to a contradiction to the assumption. Analogously, in the case that $\mathcal{I}(T[s]) < |s|$ a contradiction follows. Now, the case that $Q[s]$ and $T[s]$ differ in the times when they visit the vertices. Because of the assumption that a trajectory does not stay at the same vertex in two consecutive time intervals, there is an intersection of time intervals in which $Q[s]$ and $T[s]$ visit different vertices $u$ and $v$. Due to the strictly positive edge weights it is $e^{-d(u, v)} < 1$. This leads again to a contradiction.

For the computation of the similarity the shortest paths distance between the vertices of the graph is needed. These distances can be globally precomputed or computed on-the-fly during the computation of the similarity only for vertices $u$ that are visited by the query trajectory $Q$.

**Theorem 3.1.** Let $Q$ and $T$ be trajectories, and $s$ a time interval, the computation of the similarity $\text{Sim}(Q, T, s)$ takes $O(|Q| + |T|)$ time, if the shortest path distance $d(u, v)$ between $u, v \in V$ can be obtained in constant time.

**Proof.** Consider the query trajectory $Q = ((u_1, a_1), \ldots, (u_i, [a_i, b_i]), \ldots, (u_k, s_k))$ and the trajectory $T = ((v_1, t_1), \ldots, (v_j, [c_j, d_j]), \ldots, (v_l, t_l))$. We start the computation with $i = 1$ and $j = 1$, and $|s \cap t_i \cap s_j|$ is either zero or larger than zero. In the first case we can increase both $i$ and $j$. In the second case, we increase $i$ if $b_i < d_j$ or $j = l$, and we increase $j$ if $b_i > d_j$ or $i = k$. We repeat this for maximal $|Q| + |T|$ times and find all pairs $(u_i, s_i)$ and $(v_j, t_j)$ that have non-empty intersection. $\square$

Next, we define a distance function based on the similarity and show useful properties for it.

**Definition 3.** Let $Q$, $T$ and $R$ be trajectories, and $s$ a time interval with $s \subseteq \mathcal{I}(Q)$. We define the distance $d_S(Q, T, s) = 1 - \text{Sim}(Q, T, s)$.

Using this definition of the distance we can show that a specific type of triangle inequality holds under reasonable conditions on the time intervals of the query.

**Lemma 3.2.** Let $Q$, $T$ and $R$ be trajectories, and $s$ a time interval. If $s \subseteq \mathcal{I}(Q)$, then $d_S(Q, R, s) \leq d_S(Q, R, s) + d_S(R, T, s)$.

**Proof.** Let $t = \mathcal{I}(T)$ and $r = \mathcal{I}(R)$. We can assume without loss of generality that $s = \mathcal{I}(Q)$. We show that $1 - \text{Sim}(Q, T, s) \leq 1 - \text{Sim}(Q, R, s) + 1 - \text{Sim}(R, T, s)$. This is equivalent to $1 + \text{Sim}(Q, T, s) \geq \text{Sim}(Q, R, s) + \text{Sim}(R, T, s)$. By substituting Definition 2 and multiplying both sides with $|s|$ we get

$$|s| + \sum_{(u_i, s_i) \in Q \cap T} |s \cap s_j \cap t_i| \cdot e^{-d(u_j, v_i)} \geq$$

$$\sum_{(u_i, s_i) \in Q \cap R} |s \cap s_j \cap r_k| \cdot e^{-d(u_j, w_k)}$$

$$+ \sum_{(w_k, r_k) \in R \cap T} |s \cap r_k \cap t_i| \cdot e^{-d(w_k, v_i)}$$

Now, using the fact that

$$|s \cap t| = \sum_{(u_i, s_i) \in Q \cap T} |s \cap t_i \cap s_j|$$

...
we can rewrite the above equivalently as
\[ |s| - |s \cap t| + \sum_{(u_j,v_j) \in Q} |s \cap s_j \cap t_i| \cdot (1 + e^{-d(u_j,v_j)}) \geq \sum_{(u_j,v_j) \in Q} |s \cap s_j \cap r_k| \cdot e^{-d(u_j,w_k)} + \sum_{(w_k,v_i) \in R} |s \cap r_k \cap t_i| \cdot e^{-d(w_k,v_i)}. \]

We now want to show that the above inequality holds true. Consider the following multisets of vertex pairs. \( A \) contains the pairs \((u_j, v_j)\) that are summed up on the left side of the inequality for which \(|s \cap s_j \cap t_i| > 0\), where each \((u_j, v_i)\) is in \( A \) exactly \(|s \cap s_j \cap t_i|\) times. Similarly, \( B \) contains the pairs \((u_j, w_k)\) that are summed up during the first summation on right side of the inequality for which \(|s \cap s_j \cap r_k| > 0\), where each \((u_j, w_k)\) is in \( B \) exactly \(|s \cap s_j \cap r_k|\) times. And finally, \( C \) contains the pairs \((w_k, v_i)\) that are summed up during the second summation on right side of the inequality for which \(|s \cap r_k \cap t_i| > 0\), where each \((w_k, v_i)\) is in \( C \) exactly \(|s \cap r_k \cap t_i|\) times. Then, we show
\[
|s| - |s \cap t| + \sum_{(u_j,v_j) \in A} (1 + e^{-d(u_j,v_i)}) \geq \sum_{(u_j,w_k) \in B} e^{-d(u_j,w_k)} + \sum_{(w_k,v_i) \in C} e^{-d(w_k,v_i)}.
\]

We show that the multisets \( A, B \) and \( C \) contain vertex pairs such that the inequality holds. And let \( p \subseteq s \) be an interval of length one. For each possible \( p \) we have some vertex pairs in the multisets. We need consider the following cases (see also Figure 1):

1. \( p \cap t \neq \emptyset \) and \( p \cap r \neq \emptyset \): \( A \) contains vertex pairs \((u,v)\) but neither \( B \) nor \( C \) contain corresponding pairs. Therefore, favoring the left side of eq. (3.1).

2. \( p \cap t \neq \emptyset \) and \( p \cap r = \emptyset \): There are \((v_i,u_j)\) in \( A \), \((v_i,w_k)\) in \( B \) and \((w_k,u_j)\) in \( C \). In this case it holds that \( 1 + e^{-d(u_j,v_i)} \geq e^{-d(u_j,w_k)} + e^{-d(w_k,v_i)} \).

3. \( p \cap t = \emptyset \) and \( p \cap r \neq \emptyset \): There are no corresponding vertex pairs in \( A \) and \( C \) but in \( B \). However, this can only be the case for \(|s| - |s \cap t|\) pairs and each contributes max. 1 to the right side.

4. \( p \cap t = \emptyset \) and \( p \cap r = \emptyset \): There are no corresponding vertex pairs in \( A, B \) or \( C \).

Next, we show a strong relationship between the distance of two trajectories with respect to two different time intervals.

**Lemma 3.3.** Let \( Q \) and \( T \) be trajectories, and \( s \) and \( t \) time intervals with \( t \subseteq s \). It holds that\( d_S(Q,T,t) = |\frac{|s|}{|t|} d_S(Q,T,s)\).

**Proof.** Assuming \( s \subseteq t \) and using Definition 2 it holds that
\[
d_S(Q,T,t) = \frac{1}{|t|} \sum_{(u_j,v_i) \in Q} |t \cap s_j \cap t_i| \cdot e^{-d(u_j,v_i)} = \frac{1}{|t|} \sum_{(u_j,v_i) \in Q} |s \cap s_j \cap t_i| \cdot e^{-d(u_j,v_i)}
\]
since \( s_j \subseteq s \subseteq t \) for all \( s_j \). Now we can apply Definition 2 again and obtain
\[
d_S(Q,T,t) = |\frac{|s|}{|t|} d_S(Q,T,s)
\]

4 **Indexing the Trajectories**

In this section, we introduce our efficient indexing method for the trajectories and the application of temporal and spatial filters. We give first a high level view on how the approaches work: (1) Given a set of trajectories \( T \) a preprocessing phase constructs the index \( D \) that allows efficient queries. (2) Our approaches use the following procedure for querying given a query trajectory \( Q \) and a time interval \( s \). We compare the trajectory sim, which is computed by the distance of two trajectories with respect to two different time intervals. Note that it is possible that the candidate set of trajectories \( D \) contains all trajectories stored in \( D \). In this case we achieve small candidate sets if possible. To this end, use filters for the temporal and the spatial domain.
4.1 Pivot-Based Spatial Filters We choose $h \in \mathbb{N}$ vertices $p_1, \ldots, p_h$ from which we construct $h$ pivot trajectories $P_1, \ldots, P_h$. The $h$ vertices are the ones that are visited the most often by trajectories, where we also count multiple visits from a trajectory $T$ at a vertex. Each pivot trajectory $P_i$ stays stationary at vertex $p_i$ during the time interval $t = [a, b]$, where $a$ is the earliest starting and $b$ the latest ending time over all trajectories $T \in \mathcal{T}$. Next we compute the pairwise distance $d_S(T, P_i, t)$ between all $T \in \mathcal{T}$ and $P_i$ for $1 \leq i \leq h$ and store these distances together with the pivot trajectories.

Given a query $(Q, s)$ we compute $d_S(Q, P_i, t)$ for $1 \leq i \leq h$. Using Lemma 3.2 and Lemma 3.3 and since $|\mathcal{T}| \leq 1$, it follows that

$$|d_S(Q, P_i, t) - d_S(P_i, T, t)| \leq d_S(Q, T, t) \leq d_S(Q, T, s),$$

where we use that $t \subseteq \mathcal{I}(P_i)$ which holds by construction of $P_i$. We can use the above bound to filter out a lot of trajectories from the candidate set that are too far away from the query. To this end, we use a threshold radius $r$ such that we only keep trajectories for which

$$|d_S(Q, P_i, t) - d_S(P_i, T, t)| \leq r$$

for all $P_i$ and $1 \leq i \leq h$.

The construction of an index utilizing pivot-based spatial filtering is efficient, because we only need not compute the distance between the $h$ pivot trajectories and all $T \in \mathcal{T}$, each in $O(|\mathcal{T}| + |P_i|)$.

**THEOREM 4.1.** The index for pivot-based spatial filters can be computed in $O(|\mathcal{T}| \cdot h \cdot m)$ time, where $m$ is the maximal length of a trajectory over $\mathcal{T}$. Furthermore, it needs $O(|\mathcal{T}| \cdot h)$ memory.

4.2 Temporal Filter Notice that the candidate set $\mathcal{C} \subseteq \mathcal{T}$ only needs to contain trajectories that (partly) overlap the query time interval $s$. To filter out all trajectories that do not intersect the query interval $s$ we construct a binary interval tree using the following procedure. For each node $h$ in the tree, we have set of trajectories $\mathcal{T}_h$. We compute the median $m$ of the endpoints in $\mathcal{T}_h$ and assign all trajectories that end before $m$ to the left child and all trajectories that start after $m$ to the right child of $h$. All other trajectories are stored at $h$. We proceed recursively until we reach a minimum size for the trajectory set $\mathcal{T}_l$, where $l$ is a leaf of the tree.

We combine the temporal filter with the pivot-based spatial filter by using an interval tree that uses a pivot-based spatial filter at each node $h$ for the trajectories stored at node $h$.

4.3 Upper Bounding During the computations of the similarities between $Q$ and each trajectory $T$ in a set of trajectories $\mathcal{T}$ we can apply the following upper bounding technique. After calculating the similarity of the first $k$ trajectories, we can stop the similarity computation between $Q$ and $T$ early if we can assure that $Sim(Q, T', s) \leq l$, where $l$ is the smallest similarity between $Q$ and any $T \in \mathcal{T}$ found so far.

Let $T_1, \ldots, T_{|\mathcal{C}|}$ be the trajectories of the candidate set in order of processing. Consider the computation of the similarity $Sim(Q, T_h, s)$ for $h > k$ described in the proof of Theorem 3.1. At each step, before increasing $i$ or $j$, we obtain an upper bound $\bar{s}$ for $Sim(Q, T_h, s)$ by assuming each remaining time step the trajectories are at the same vertices. If $\bar{s}$ is smaller than the $k$ lowest similarity found so far, we stop the computation of $Sim(Q, T_h, s)$ and proceed with $Sim(Q, T_{h+1}, s)$.

5 Comparison to Existing Approaches

Grossi et al. [6] introduce three algorithms answering top-$k$ similarity queries in a spatio-temporal setting. The idea of their baseline algorithm is to add a preprocessing phase in which an interval tree is constructed at each vertex $v$ of the graph. This interval tree contains all pairs of $(T, id, t)$ in the case that a trajectory $T \in \mathcal{T}$ visits $v$ or any of its adjacent vertices during time interval $t$, where $T, id$ is the identifier of the trajectory $T$. Then, using the constructed index, a query $(Q, s)$ is answered by visiting all vertices $v$ with $(v, t) \in Q$ and collecting all ids of trajectories that visit the vertex or any of its neighbors during $t \cap s$. With the collected set of ids the candidate set of trajectories can be evaluated and the top-$k$ similar trajectories are found. The running time and memory requirements therefore depend on the maximal vertex degree. Moreover, the algorithm solves a special case, in which only trajectories are considered that have at least one vertex in hop-distance one to a vertex of the query trajectory.

A simple example for which the algorithm will fail to find a similar trajectory can be constructed in a graph consisting of a chain of four vertices, i.e., $G = \{v_1, \ldots, v_4\}$, $(v_1, v_2, v_2 v_3, v_3 v_4)$, and a trajectories $T = ((v_1, [0, 1]))$ and $Q = ((v_4, [0, 1]))$. After the preprocessing phase only the interval trees at $v_1$ and $v_2$ contain the id of $T$. For a query $(Q, [0, 1])$ the algorithm will only look at the empty interval tree at $v_4$ and cannot find $T$.

Grossi et al. [6] also introduce two heuristic approaches for the top-$k$ query problem. Their idea is to reduce the graph size and shrink the length of the query trajectory or all trajectories, in order to save time by reducing the number of distance computations between vertices. However, this often leads to larger candidate
sets, and hence more evaluations of the similarity function are necessary.

Our approaches differ from the ones suggested by Grossi et al. \cite{6} as follows. We have introduced an alternative similarity function for which we could show certain metric properties. Our indices attempt to reduce the size of the trajectory candidate set and therefore reduce the number of similarity computations. Furthermore, by using the upper bounding technique without preprocessing and constructing an index we obtain an exact algorithm that is competitive in running time.

6 Experiments

In this section, we evaluate our new algorithms and compare them to the approaches suggested in \cite{6}. We are interested in answering the following questions:

Q1: How fast are the indexing times of our algorithms compared to the baseline and to the heuristic approaches in \cite{6}?

Q2: How fast are queries of our algorithms compared to the baseline and to the heuristic approaches in \cite{6}? Do our index approaches accelerate the queries?

Q3: How good is the quality of the provided approximated top-k queries?

6.1 Algorithms and Experimental Protocol

We implemented the following new algorithms:

- **Exact** is the direct and exact algorithm that does not use indexing. It iteratively computes the similarity between the query and all trajectories using the upper bounding described in section 4.3.

- **Tree** is the index that uses an interval tree with additional pivot-based spatial filtering at each node of the interval tree.

- **Pivot** is the index that applies the pivot-based spatial filtering globally.

**Exact**, **Tree**, and **Pivot** use the upper bounding technique (Section 4.3). Furthermore, we implemented the following algorithms from \cite{6}.

- **Gbase**, the baseline algorithm in \cite{6} (see section 5).

- **Gshq** and **GshqT** denote their heuristic approaches based on shrinking the graph and the trajectories and gaining advantage of the smaller graph size and reduced trajectory lengths (see section 6).

All of these algorithms are based on our new similarity measure. We implemented all algorithms in C++ using GNU CC Compiler 9.3.0 with the flag --O2. All experiments were conducted on a workstation with an AMD EPYC 7402P 24-Core Processor with 2.80 GHz and 256 GB of RAM running Ubuntu 18.04.3 LTS using a single core.

6.2 Data Sets

For the evaluation of the algorithms we used the following data sets.

- **Facebook 1 & 2**: The network consists of Facebook friendship relations \cite{9} and is provided by the Stanford Network Analyses Project\footnote{https://snap.stanford.edu/data/ego-Facebook.html}. We have generated synthetic trajectories.

- **Milan**: The Milan data set is based on GPS trajectories of private cars in the city of Milan\footnote{https://sobigdata.d4science.org/catalogue-sobigdata?path=/dataset/gps_track_milan_italy}.

- **T-Drive**: The data set contains GPS data of taxi trajectories in Beijing \cite{14, 15}.

For the **Milan** and **T-Drive** data set we have generated a network by first interpreting each GPS location point as a vertex and then clustering these vertices using the k-means algorithm. The resulting clusters are the final vertices. Two clusters are connected by an edge if at least one trajectory visits a vertex in each of both clusters in a consecutive time interval. We assign the great-circle distance between the centers of the clusters as distance to the edge. Table 1 shows some statistics for the data sets.

6.3 Results

We answer questions Q1 to Q3.

Q1: Table 2 shows the running times for indexing the data sets. For **Tree** and **Pivot**, we choose the number of pivots $h = 10$ for all data sets but **Milan**, for which we set $h = 20$. The construction of our index structures is orders of magnitude faster than that of the approaches suggested in \cite{6}. The largest speed-up has been achieved
for the Facebook1 data sets, for which Pivot is over 6000 times faster. Out of all indexing approaches, as expected, Pivot is the fastest method for all data sets.

Table 2: Indexing times in seconds.

| Algorithm | Data set | TREE | Pivot | GBASE | GSHQ | GSHQT |
|-----------|----------|------|-------|-------|------|-------|
| Facebook1 | 0.61     | 0.48 | 2.965.24 | 342.74 | 248.04 |
| Facebook2 | 6.49     | 4.97 | 3.921.03 | 442.55 | 327.28 |
| Milan     | 1.25     | 0.68 | 193.77  | 59.26  | 52.91 |
| T-Drive   | 1.26     | 0.91 | 1.509.30 | 135.68 | 87.86 |

Q2: We selected 100 trajectories randomly from the data sets as queries. The query interval is set to \( I(Q) \). We ran the algorithms for \( k = 2^i \) with \( 0 \leq i \leq 6 \). Table 3 shows the threshold radii that we used for the pivot-based filtering, and Table 4 shows the average running times for querying a trajectory from the data set.

Table 3: Threshold values \( r \) used for the pivot based filters during query time.

| Data set | Index | Facebook1 | Facebook2 | Milan | T-Drive |
|----------|-------|-----------|-----------|-------|---------|
| TREE     | 0.35  | 0.25      | 0.20      | 0.60  |
| Pivot    | 0.15  | 0.15      | 0.005     | 0.30  |

First note that the query times of our exact approach is about the same as that for GBASE. For the Facebook2 and the T-Drive instances GBASE is up 0.2 seconds slower. Note that the running time of GBASE does not depend on \( k \), since the collected candidate set is the same for all \( k \). The latter is true for our exact approach, however, the upper bounding leads to varying running times. We will see later that GBASE in contrast to our exact approach does not always find the optimal solution set. Both of our index structures lead to accelerated query times compared to the exact approach. The largest speed-up of about 3 to 4 has been achieved for TREE on the Facebook2 instances. Note that almost always the two heuristic approaches GSHQ and GSHQT are much slower in answering the queries. For the Milan and T-Drive instances they are even slower than our exact approach.

Q3: In order to evaluate the quality of our query results, we use the similarity score ratio (SSR) defined in [6]. The SSR of two sets \( T_1 \) and \( T_2 \) of trajectories with respect to a query is defined as \( SSR(T_1, T_2, (Q, s)) = \frac{\sum_{t_1 \in T_1} Sim(Q, t_1, s)}{\sum_{t_2 \in T_2} Sim(Q, t_2, s)} \). We compare the results of the indices to the results of the exact algorithm EXACT.

Table 4: Query times in seconds for top-k similarity queries. The running times are the average and standard deviations over 100 queries.

| Algorithm | Data set | k     | Exact | TREE | Pivot | GBASE | GSHQ | GSHQT |
|-----------|----------|-------|-------|------|-------|-------|------|-------|
| Facebook1 | 1        | 0.22   | 0.20  | 0.13 | 0.22  | 0.13  | 0.15  | 0.15  |
| Facebook1 | 2        | 2.30   | 1.54  | 1.45 | 2.57  | 1.53  | 2.55  | 2.77  |
| Milan     | 1        | 0.02   | 0.03  | 0.01 | 0.01  | 0.06  | 0.05  | 0.04  |
| T-Drive   | 1        | 0.29   | 0.28  | 0.25 | 0.44  | 0.20  | 0.69  | 0.37  | 0.64 |
| Facebook1 | 2        | 0.22   | 0.17  | 0.14 | 0.23  | 0.13  | 0.15  | 0.15  |
| Facebook1 | 2        | 2.60   | 1.82  | 1.46 | 2.58  | 1.54  | 2.58  | 2.85  |
| Milan     | 2        | 0.02   | 0.03  | 0.01 | 0.01  | 0.06  | 0.05  | 0.04  |
| T-Drive   | 2        | 0.30   | 0.18  | 0.25 | 0.45  | 0.71  | 0.65  | 0.30  |
| Facebook1 | 4        | 0.22   | 0.17  | 0.13 | 0.22  | 0.13  | 0.15  | 0.15  |
| Facebook1 | 4        | 2.11   | 0.82  | 1.46 | 2.58  | 1.54  | 2.58  | 2.85  |
| Milan     | 4        | 0.02   | 0.03  | 0.02 | 0.01  | 0.06  | 0.05  | 0.04  |
| T-Drive   | 4        | 0.31   | 0.19  | 0.26 | 0.45  | 0.71  | 0.65  | 0.30  |
| Facebook1 | 8        | 0.22   | 0.17  | 0.13 | 0.22  | 0.13  | 0.15  | 0.15  |
| Facebook1 | 8        | 2.43   | 0.83  | 1.47 | 2.58  | 1.54  | 2.58  | 2.85  |
| Milan     | 8        | 0.03   | 0.03  | 0.01 | 0.01  | 0.06  | 0.05  | 0.04  |
| T-Drive   | 8        | 0.31   | 0.19  | 0.26 | 0.45  | 0.71  | 0.65  | 0.30  |
| Facebook1 | 16       | 0.22   | 0.17  | 0.13 | 0.22  | 0.13  | 0.15  | 0.15  |
| Facebook1 | 16       | 2.44   | 0.84  | 1.49 | 2.58  | 1.54  | 2.58  | 2.85  |
| Milan     | 16       | 0.03   | 0.03  | 0.02 | 0.01  | 0.06  | 0.05  | 0.04  |
| T-Drive   | 16       | 0.32   | 0.19  | 0.27 | 0.45  | 0.72  | 0.65  | 0.30  |
| Facebook1 | 32       | 0.22   | 0.17  | 0.13 | 0.22  | 0.13  | 0.15  | 0.15  |
| Facebook1 | 32       | 2.46   | 0.84  | 1.49 | 2.58  | 1.54  | 2.58  | 2.85  |
| Milan     | 32       | 0.03   | 0.03  | 0.02 | 0.01  | 0.06  | 0.05  | 0.04  |
| T-Drive   | 32       | 0.32   | 0.19  | 0.27 | 0.45  | 0.72  | 0.65  | 0.30  |
| Facebook1 | 64       | 0.22   | 0.17  | 0.13 | 0.22  | 0.13  | 0.15  | 0.15  |
| Facebook1 | 64       | 2.48   | 0.85  | 1.49 | 2.58  | 1.54  | 2.58  | 2.85  |
| Milan     | 64       | 0.03   | 0.03  | 0.02 | 0.01  | 0.06  | 0.05  | 0.04  |
| T-Drive   | 64       | 0.33   | 0.20  | 0.27 | 0.45  | 0.72  | 0.65  | 0.30  |

Table 5 shows the average SSR values and the standard deviations over 100 queries.

First we observe that as expected (see section 4) the baseline GBASE [6] has not always found the optimal solution set. The SSR score takes values of 0.85 for \( k = 64 \) on the Milan data set. With increasing value of \( k \) the SSR value for our TREE approach decreases from 0.94 for \( k = 1 \) to 0.77 for \( k = 64 \). However, for our Pivot approach the decrease is less strong; the SSR score is always above 0.92 for the instances Facebook1, Facebook2, and T-Drive. For the MILAN instances our heuristic approaches do not behave very well. Here, the SSR score for both, TREE and Pivot takes values below 0.8 for \( k \geq 16 \). We believe that the reason for these low values is the small value of \( r \) chosen in our experiments. However, a larger value of \( r \) will not lead to an improvement of the running time with respect to exact computations, which are quite fast. This is because of the length of the Milan trajectories is relatively small (only about 33, see Table 1). For the Facebook2 instances the SSR score of Pivot is always above 0.99. Surprisingly, the values of the GSHQ and GSHQT heuristics for Facebook1, Facebook2, and T-Drive are always above 0.9. However, remember that their query time takes longer than that for TREE and Pivot – for T-Drive even more than twice as long as our exact computations.
Moreover, we suggest novel spatial and temporal index-basis for an upper bounding procedure which accelerates the searching trajectories by locations: An efficiency study. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD ’10, page 2537-266, New York, NY, USA, 2010. Association for Computing Machinery.

7 Conclusion

We studied the problem of computing the top-k most similar trajectories in a graph to a given query trajectory. For this, we propose a new spatio-temporal similarity measure based on work of Grossi et al. From this, we derive a distance function which satisfies a triangle inequality under certain conditions. This builds the basis for an upper bounding procedure which accelerates finding exact solutions of top-k trajectory queries. Moreover, we suggest novel spatial and temporal indexing structures that further accelerate the search process; however, our experimental evaluations have shown that for certain instances (e.g., the Milan data set) for which the computation is already fast, this acceleration is at the expense of the solution quality.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In D. B. Lomet, editor, Foundations of Data Organization and Algorithms, pages 69–84, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[2] L. Chen, M. T. ¨Ozsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, SIGMOD ’05, page 491–502, New York, NY, USA, 2005. Association for Computing Machinery.

[3] L. Chen, S. Shang, B. Yao, and K. Zheng. Spatio-temporal top-k term search over sliding window. World Wide Web, 22(5):1953-1970, 2019.

[4] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie. Searching trajectories by locations: An efficiency study. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD ’10, page 2537-266, New York, NY, USA, 2010. Association for Computing Machinery.

[5] A. Driemel, I. Psarros, and M. Schmidt. Sublinear data structures for short Frechet queries. CoRR, abs/1907.04420, 2019.

[6] R. Grossi, A. Marino, and S. Moghasedi. Finding structurally and temporally similar trajectories in graphs. In 18th International Symposium on Experimental Algorithms (SEA 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[7] J.-R. Hwang, H.-Y. Kang, and K.-J. Li. Searching for similar trajectories on road networks using spatio-temporal similarity. In Y. Manolopoulos, J. Pokorny, and T. K. Sellis, editors, Advances in Databases and Information Systems, pages 282–295, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[8] P. Indyk. Approximate nearest neighbor algorithms for frechet distance via product metrics. In Proceedings of the 18th Annual Symposium on Computational Geometry, Barcelona, Spain, June 5-7, 2002, pages 102–106, 2002.

[9] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, pages 548–556, 2012.

[10] S. Shang, R. Ding, K. Zheng, C. S. Jensen, P. Kalnis, and X. Zhou. Personalized trajectory matching in spatial networks. VLDB J., 23(3):449–468, 2014.

[11] E. Tiaskas, A. N. Papadopoulos, A. Nanopoulos, Y. Manolopoulos, D. Stojanovic, and S. Djardevic-Kajan. Trajectory similarity search in spatial networks. In 2006 10th International Database Engineering and Applications Symposium (IDEAS’06), pages 185–192. IEEE, 2006.

[12] E. Tiaskas and D. Rafailidis. Scalable trajectory similarity search based on locations in spatial networks. In L. Bellatreche and Y. Manolopoulos, editors, Model and Data Engineering - 5th International Conference, MEDI 2015, Rhodes, Greece, September 26-28, 2015, Proceedings, volume 9344 of Lecture Notes in Computer Science, pages 213–224. Springer, 2015.

[13] Y. Xia, G.-Y. Wang, X. Zhang, G.-B. Kim, and H.-Y. Bae. Spatio-temporal similarity measure for network constrained trajectory data. International Journal of Computational Intelligence Systems, 4:1070–1079.
[14] J. Yuan, Y. Zheng, X. Xie, and G. Sun. Driving with knowledge from the physical world. In C. Apté, J. Ghosh, and P. Smyth, editors, Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011, pages 316–324. ACM, 2011.

[15] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In D. Agrawal, P. Zhang, A. E. Abbadi, and M. F. Mokbel, editors, 18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2010, November 3-5, 2010, San Jose, CA, USA, Proceedings, pages 99–108. ACM, 2010.