Fast Subtrajectory Similarity Search in Road Networks under Weighted Edit Distance Constraints

Satoshi Koide
Toyota Central R&D Labs.
koide@mosk.tytlabs.co.jp

Chuan Xiao
Osaka University and Nagoya University
chuanx@nagoya-u.jp

Yoshiharu Ishikawa
Nagoya University
ishikawa@i.nagoya-u.ac.jp

ABSTRACT

In this paper, we address a similarity search problem for spatial trajectories in road networks. In particular, we focus on the *subtrajectory similarity search* problem, which involves finding in a database the subtrajectories similar to a query trajectory. A key feature of our approach is that we do not focus on a specific similarity function; instead, we consider *weighted edit distance* (WED), a class of similarity functions which allows user-defined cost functions and hence includes several important similarity functions such as EDR and ERP. We model trajectories as strings, and propose a generic solution which is able to deal with any similarity function belonging to the class of WED. By employing the filter-and-verify strategy, we introduce *subsequence filtering* to efficiently prunes trajectories and find candidates. In order to choose a proper subsequence to optimize the candidate number, we model the choice as a discrete optimization problem (NP-hard) and compute it using a 2-approximation algorithm. To verify candidates, we design *bidirectional tries*, with which the verification starts from promising positions and leverage the shared segments of trajectories and the sparsity of road networks for speed-up. Experiments are conducted on large datasets to demonstrate the effectiveness of WED and the efficiency of our method for various similarity functions under WED.

PVLDB Reference Format:
Satoshi Koide, Chuan Xiao, Yoshiharu Ishikawa. Fast Subtrajectory Similarity Search in Road Networks under Weighted Edit Distance Constraints. *PVLDB*, 13(11): 2188-2201, 2020.
DOI: https://doi.org/10.14778/3407790.3407818

1. INTRODUCTION

Vehicular transportation is facing a crucial turning point as data-driven information technology advances. Data-driven approaches, such as intelligent routing and ride-sharing, are expected to resolve important social issues, such as environmental problem and traffic congestion; they are therefore actively studied in many fields, including database research. Accordingly, fundamental operations, such as indexing and retrieval, on huge vehicular trajectory data are becoming increasingly important [19, 20, 22, 23, 41, 47, 63].

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407818

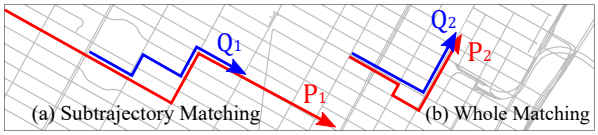

Figure 1: Subtrajectory matching/Whole matching.

This paper addresses the similarity search problem over trajectories in road networks, which is a classical but still active area of spatial database research [41, 47, 54, 55, 63]. Unlike most existing studies that target *whole matching* between data and query trajectories (i.e., the entire trajectories are similar), we tackle the *subtrajectory* similarity search problem, which finds in a vehicular trajectory database the subtrajectories similar to a query (Figure 1).

Why subtrajectory similarity search? One motivating application is travel time estimation along a given path. Recent on-the-fly approaches [51, 52] estimate the travel time distribution by retrieving historical trajectories that contain the query path as a subtrajectory immediately after the query arrived. Other applications include alternative route suggestion that finds the variations of a query path in the database as alternative routes and path popularity estimation [8, 20, 29] that counts the frequency of appearance of a given path in the database as a subtrajectory.

Exact path queries have been studied for subtrajectory search [20, 23]; however, exact path queries only find trajectories containing a subtrajectory that exactly matches a query trajectory. Hence similarity queries are adopted to retrieve more semantically relevant results for various applications [6, 7, 17, 39, 41, 44, 47, 57, 62, 63]. For example, similarity queries can handle the errors caused by sampling strategies, spatial transformations, or natural noises [6, 7, 44, 47, 63], which are common in real data. In addition, travel time estimation suffers from *data sparsity* (i.e., there are few historical trajectories that exactly travel a query path in the specified time slot) even in urban areas. Subtrajectory similarity queries have been used to address this issue; e.g., by path similarity [16] or road segment similarity (types, number of lanes, etc.) [50], based on the observation that paths or road segments with similar contexts may have similar travel time.

Trajectory similarity functions. To measure the similarity between trajectories, a similarity function must be selected. Albeit many trajectory similarity functions have been proposed [6, 7, 11, 14, 39, 40, 42, 45, 47, 54, 55, 57, 62, 63], as demonstrated experimentally [6, 7, 47, 63]), every similarity function has its

own advantages and disadvantages. In other words, there is no "best" trajectory similarity function, and the choice depends on the application scenario. Accordingly, for trajectory similarity search, general methodologies that do not depend on a specific similarity function are preferable.

In this paper, we consider *weighted edit distance* (WED), a class of similarity functions that includes several ones commonly used in trajectory analysis, such as edit distance on real sequences (EDR) [7] and edit distance with real penalty (ERP) [6], which have been shown capable of handling different sampling strategies, spatial transformations, and/or noises, and better than other functions such as dynamic time warping (DTW) and longest common subsequence (LCSS) [44]. WED is *flexible* in the sense that it allows user-defined cost functions. As such, it not only covers EDR and ERP but also their extensions (e.g., the adaptations using road network distance instead of binary or Euclidean distance). WED also captures the semantics of the aforementioned similarities for travel time estimation [16, 50], and is able to measure the road segments that differ between two trajectories, thereby (reversely) expressing semantics similar to longest overlapping road segments (LORS) [47] and longest common road segments (LCRS) [63].

Challenges. For real applications like trajectory analysis [41, 63], it is desirable to return query results in seconds or even less. Trajectories in road networks can be modeled as strings, and the problem is converted to substring search. A naive approach is scanning the database using the Smith-Waterman algorithm [43]; however, this is inefficient for large datasets as it does not employ indexing. For fast query processing, most string similarity algorithms resort to the *filter-and-verify* paradigm, which finds a set of candidates and then verify them. The most widely used is q-gram filtering [10, 13, 37, 53, 56] (also for trajectories [7]). However, it targets Levenshtein distance (unit cost) and does not deliver efficient performance for WED, because the lower bound of common q-gram number become very loose (even ≤ 0 and thus useless) when substitution cost is arbitrarily small (e.g., ERP). Partition-based [25, 49] and trie-based [12] methods are not applicable either, because it is hard to derive the partition size for WED, and they are designed for whole matching. Another key property resides in the sparsity of road networks (i.e., the alphabet is very large but spatially restricted), which has not been exploited in these solutions. Another line of work is trajectory similarity methods [6,41,47,63]. However, these techniques are designed for whole matching and become inefficient or inapplicable on subtrajectories. Many of them rely on their own similarity functions (e.g., the ERP-index [6] exploits the triangle inequality) or need adaptations to switch between functions (e.g., DITA [41] requires a pivoting strategy depending on the similarity function). Moreover, all the aforementioned methods bear no performance guarantee on candidate size. Seeing these challenges, we aim to design a *unified* yet *efficient* algorithm applicable to a wide range of similarity functions under the class of WED.

Contributions. Our contributions are as follows:

- We propose the first indexing and retrieval method for trajectories in road networks that supports subtrajectry search on WED. The query processing algorithm does not depend on a specific similarity function; it supports any user-specified edit distance with a **unified** and exact algorithm such that there is no need to adapt the algorithm to switch between similarity functions. To tackle the efficiency issue, we model trajectories as strings and follow the filter-and-verify paradigm.
- To generate candidates, we propose the **subsequence filtering** (§ 3), such that any trajectory in the result set must share at least one element, or a neighbor (in terms of the cost functions) of the element, with a chosen subsequence of the query. In order to choose a subsequence to optimize the candidate size, we model this as a discrete optimization problem and show its *NP-hardness* as well as a polynomial-time *2-approximation* algorithm. We also give the condition under which this algorithm finds the *optimal* subsequence. Indexing and search algorithms (§ 4) are devised based on this filtering strategy.
- To verify candidates (§ 5), we start with the positions at which candidates are found and develop pruning techniques for a **local verification** algorithm, so that only the promising part of each candidate is computed for WED. We share computation for the common subtrajectories of candidates, and design **bidirectional tries** to cache such computation by exploiting the sparsity of road networks which leads to *low cache miss rate*.
- We conducted extensive experiments on large real datasets (§ 6). The results demonstrate the effectiveness of WED and the efficiency of our method on various similarity functions under the class of WED, as well as the effectiveness of the components in the proposed solution.

2. PRELIMINARIES

2.1 Framework and Data Model

We assume trajectories are constrained in a road network. A query essentially consists of three components: (i) a trajectory, (ii) a distance function, and (iii) a distance threshold. On receiving a query, we find the data trajectories that approximately (satisfying the distance constraint) contains the query as a subtrajectory. In this paper, we assume datasets and indexes are stored in main memory.

The road network is modeled as a directed graph $G = (V, E)$. Each vertex $v \in V$ is associated with its coordinate in $\mathbb{R}^2$. Each edge $e \in E$ is associated with a weight (e.g., travel time or distance in a road network) denoted by $w(e)$.

A trajectory is modeled as a path on G, i.e., a consecutive sequence $v_1 v_2 \cdots v_n$ of vertices. We refer to this trajectory representation as *vertex representation*. Equivalently, the path can be represented by the corresponding *edge representation* $e_1 e_2 \cdots e_{n-1}$, where $e_i = (v_i, v_{i+1}) \in E$. These representations can be converted from raw trajectory (a sequence of spatial coordinates) through map matching (we employ the HMM map matching for this purpose [35]). The techniques proposed in this paper can support both representations.

Timestamps are associated with each trajectory. Following the trajectory models in existing studies (e.g., [20, 23]), we assume that timestamps are recorded at each vertex. In summary, we employ the following definition of trajectories:

Definition 1 (Trajectory) *A trajectory is a tuple (P, T), where P is a path on G, and T is a sequence of timestamps associated with each vertex in P.*

We first focus on dealing with paths and then extend our techniques to the case with time constraints. Thus, we also denote a trajectory by its path P.

Notation. A path can be regarded as a string. We denote an alphabet set by Σ. For vertex representation, $\Sigma = V$. For edge representation, $\Sigma = E$. The set of all possible strings on

Σ is denoted by Σ^*. An empty symbol is denoted by ε, and $\Sigma^+ := \Sigma \cup \{\varepsilon\}$. Given a trajectory P, its i-th element is P_i, and a subtrajectory (substring) of A from i to j is $P_{i:j}$ (if $i > j$, $P_{i:j}$ represents an *empty string*). $|P|$ denotes the length of the trajectory (string). We say $P' \sqsubseteq P$ if P' is a *subtrajectory* of P. Similarly, $P' \subseteq P$ means that P' is a *subsequence* of P; i.e., there exist $i_1, i_2, \cdots, i_k$ such that $i_1 < i_2 < \cdots < i_k$ and $P' = P_{i_1} P_{i_2} \cdots P_{i_k}$. $\{1, 2, \cdots, n\}$ is denoted by $[\![n]\!]$.

2.2 Weighted Edit Distance

2.2.1 Concept

The Levenshtein distance, the most fundamental form of edit distance (on strings), counts the minimum number of *edit operations* needed to convert a string P into another string Q. The edit operations usually consists of *insertion*, *deletion*, and *substitution* of a symbol.

We consider a general class of edit distances where the costs of edit operations can take any values. Given two symbols $a, b \in \Sigma$, we denote the insertion, deletion and substitution costs by $\mathsf{ins}(a)$, $\mathsf{del}(b)$, and $\mathsf{sub}(a, b)$, respectively. The weighted edit distance (WED) between two trajectories $P = P_{1:m}$ and $Q = Q_{1:n}$, denoted by $\mathsf{wed}(P, Q)$, is defined recursively:

$$\mathsf{wed}(\varepsilon, Q_{1:n}) = \sum_{j=1}^{n} \mathsf{ins}(Q_j), \quad \mathsf{wed}(P_{1:m}, \varepsilon) = \sum_{i=1}^{m} \mathsf{del}(P_i),$$

$$\mathsf{wed}(P_{1:m}, Q_{1:n}) = \min \begin{cases} \mathsf{wed}(P_{1:m-1}, Q_{1:n-1}) + \mathsf{sub}(P_m, Q_n), \\ \mathsf{wed}(P_{1:m-1}, Q_{1:n}) + \mathsf{del}(P_m), \\ \mathsf{wed}(P_{1:m}, Q_{1:n-1}) + \mathsf{ins}(Q_n). \end{cases}$$

We can compute $\mathsf{wed}(P_{1:m}, Q_{1:n})$ by dynamic programming in $O(mn)$ time. Furthermore, to keep notation simple, we define $\mathsf{sub}(a, \varepsilon) := \mathsf{del}(a)$ and $\mathsf{sub}(\varepsilon, b) := \mathsf{ins}(b)$.

Assumptions. To obtain a meaningful similarity function, we make some assumptions on the edit operation costs. First, to make $\mathsf{wed}(P, Q)$ nonnegative, we assume $\mathsf{sub}(a, b) \geq 0$ for $a, b \in \Sigma^+$. To make $\mathsf{wed}(P, Q)$ symmetric, we assume $\mathsf{sub}(a, b) = \mathsf{sub}(b, a)$ (and this implies $\mathsf{ins}(a) = \mathsf{del}(a)$). Finally, to make $\mathsf{wed}(P, P) = 0$ hold, we assume $\mathsf{sub}(a, a) = 0$. Note that we do not enforce the triangle inequality $\mathsf{wed}(P, Q) \leq \mathsf{wed}(P, R) + \mathsf{wed}(R, Q)$. Also, we do not enforce $\mathsf{wed}(P, Q) = 0 \Rightarrow P = Q$, i.e., $\mathsf{wed}(P, Q) = 0$ does not mean $P = Q$.

Proposition 1 *With the assumptions above, we have:* (i) $\mathsf{wed}(P, Q) \geq 0$*, (nonnegativity);* (ii) $\mathsf{wed}(P, P) = 0$*, (pseudo-positive definite);* (iii) $\mathsf{wed}(P, Q) = \mathsf{wed}(Q, P)$*. (symmetry)*

Proposition 1 implies that WED is not metric in general. Next, we show that WED contains some existing similarity functions as special cases.

2.2.2 Known Instances of WED

Levenshtein Distance. The well-known Levenshtein distance (Lev) is obtained by setting

$$\mathsf{sub}(a, b) = \begin{cases} 0, & (a = b) \\ 1, & (a \neq b) \end{cases}, \ \mathsf{ins}(a) = 1, \ \mathsf{del}(b) = 1. \quad (1)$$

This can be used for both the vertex and edge representations.

Edit Distance on Real Sequence (EDR). EDR [7] is defined on real-valued sequences. For the data trajectory P, we use vertex representation. A query Q is not necessarily restricted on road networks. We can cover EDR by setting

$$\mathsf{sub}(a, b) = \begin{cases} 0, & (d(a, b) \leq \varepsilon) \\ 1, & (otherwise) \end{cases}, \ \mathsf{ins}(a) = 1, \ \mathsf{del}(b) = 1, \quad (2)$$

where $\varepsilon > 0$ is a predefined matching threshold. We employ Euclidean distance for $d(a, b)$. EDR is not a metric, i.e., the triangle inequality does not hold.

Edit Distance with Real Penalty (ERP). ERP [6] is also defined on real-valued sequences, obtained by setting

$$\mathsf{sub}(a, b) = d(a, b), \ \mathsf{ins}(a) = d(a, g), \ \mathsf{del}(b) = d(b, g), \quad (3)$$

where $g \in \mathbb{R}^2$ is a predefined reference point (e.g., the barycenter of the vertices in V). ERP is a metric.

2.2.3 Network-aware Similarity Functions

WED is more flexible than the aforementioned distance functions in the sense that users can define their own costs tailored to the application; e.g., for trajectory analysis in a road network, users may use a distance defined on the road network instead of Euclidean distance, or count the road segments that differ in two trajectories. Next we show some examples.

NetERP and NetEDR. A popular trajectory similarity definition employs shortest path distance between two vertices a and b [11, 14, 39, 40, 45]. By replacing Euclidean distance in EDR (Eq.(2)) and ERP (Eq.(3)) with shortest path distance, we obtain new similarity functions, referred to as NetEDR and NetERP, respectively. For directed graphs, shortest path distance is not symmetric, which violates the assumption above. One way to fix this is to make the road network undirected.

In ERP, we need a reference point; in NetERP, we use a constant insertion/deletion cost instead, $G^{(\mathsf{del})}_{\mathsf{NetERP}} > 0$, defined by users. This makes NetERP non-metric, but this does not affect our method since it does not use the triangle inequality.

Shortest Unshared Road Segments (SURS). Another idea behind several existing similarity functions is to evaluate the total edge weights (e.g., distance or travel time) that are shared (or unshared) between two trajectories [47, 54, 55, 63]. To express such semantics using WED, we define *shortest unshared road edges* (SURS) for trajectories in edge-representation:

$$\mathsf{sub}(a, b) = w(a) + w(b), \ \mathsf{ins}(a) = w(a), \ \mathsf{del}(b) = w(b), \quad (4)$$

where $w(a)$ is a given travel cost for a road edge $a \in E$. Because $\mathsf{sub}(a, b) = \mathsf{ins}(a) + \mathsf{del}(b)$, substitution is equivalent to a combination of insertion and deletion; therefore, SURS essentially counts the total travel costs of edges not shared between two trajectories, considering the order of sequence elements. Note that the functions in [47, 54, 55, 63] measure similarity, while SURS measures distance.

Example 1 *Given two paths* $P = \texttt{befg} \in \Sigma^*$ *and* $Q = \texttt{abcdg} \in \Sigma^*$ *the optimal alignment that yields the SURS is:*

$$\begin{array}{ll} \texttt{- b - - e f g} & (= P) \\ \texttt{a b c d - - g} & (= Q). \end{array}$$

$\mathsf{SURS}(P, Q)$ *is the total cost of the edges aligned to the gap symbol, i.e.,* $w(\texttt{a}) + w(\texttt{c}) + w(\texttt{d}) + w(\texttt{e}) + w(\texttt{f})$.

So far we have discussed WED instances. Furthermore, given a supervised machine learning task, we may optimize the edit operation costs of WED using a technique in [18].

2.2.4 Other Similarity Functions

There are also other similarity functions not belonging to WED. For example, in DTW, one element of a trajectory can be aligned to multiple elements in the other; this is not allowed in WED. LCSS, LORS, and LCRS are not WED either, because they are measure common subsequence rather than distance.

2.3 Problem Setting

To give a formal definition of our subtrajetory search problem, we first define the term subtrajectory matching.

Definition 2 (Subtrajectory Matching) *Given a query $Q \in \Sigma^*$ and a trajectory $P \in \Sigma^*$, we say a subtrajectory $P_{i:j}$ of P matches Q (and vice versa) iff $\mathsf{wed}(P_{i:j}, Q) < \tau$, where τ is a threshold.*

Example 2 *Consider a trajectory $P = \texttt{ABCDE}$. As $|P| = 5$, there are $|P|(|P|+1)/2 = 15$ subtrajectories. Consider a query $Q = \texttt{BFD}$ under Lev with $\tau = 2$. Then, $P_{2:4} = \texttt{BCD}$ satisfies $\mathsf{wed}(P_{2:4}, Q) = 1 < \tau$ and thus matches Q (note that we use "$<$" not "$\leq$" in the problem definition).*

We denote a set of N data trajectories by $\mathcal{T} = \{(P^{(id)}, T^{(id)}\}_{id=1}^{N}$. Our problem is defined as follows.

Definition 3 (Subtrajectory Similarity Search) *Given $(Q, \mathcal{T}, \mathsf{wed}, \tau)$, find in $\mathcal{T}$ the subtrajectories that match Q, i.e.,*

$$\mathit{SubtrajSimSearch}(Q, \mathcal{T}, \mathsf{wed}, \tau) = \{(id, s, t) \mid P_{s:t}^{(id)} \text{matches } Q\}.$$

To avoid the case that Q is similar to an empty trajectory (i.e., $\mathsf{wed}(Q, \varepsilon) < \tau$), we assume that $\sum_{q \in Q} \mathsf{ins}(q) \geq \tau$ for a meaningful problem definition.

Temporal Constraints. Some applications require consideration of temporal condition. For the on-the-fly travel time estimation mentioned in § 1, searching trajectories that traveled during a given time interval, say I, is important (e.g., rush hour). This condition can be written as $[T_i, T_j] \subseteq I$, or $[T_i, T_j] \cap I \neq \emptyset$, where i and j are the matched positions in Definition 2. Another application may require constraints on average speed (i.e., $D(P_{i:j})/(T_j - T_i)$ where D is the distance of $P_{i:j}$) or travel time for the query or some of the road segments.

A simple and general approach is checking temporal constraints *after* solving the subtrajectory similarity search. This allows us to treat *any kind of* temporal constraints. For some cases, however, speed-up can be achieved by considering temporal conditions during the filtering step, as discussed in § 4.3.

3. FILTERING PRINCIPLE

We consider designing an exact solution to subtrajectory similarity search. A naive solution is enumerating all subtrajectories of the trajectories in $\mathcal{T}$ and computing the WED to the query. The time complexity is $O(\sum_{id=1}^{N} |P^{(id)}|^3 \cdot |Q|)$. An improvement is achieved by the Smith-Waterman (SW) algorithm [43] [1]. It avoids enumerating subtrajectories and checks if Q matches some $P_{i:j}$ in $O(|P| \cdot |Q|)$ time (note that the threshold τ can be exploited for speed-up but it does not improve the time complexity), hence reducing the time complexity of processing a query to $O(\sum_{id=1}^{N} |P^{(id)}| \cdot |Q|)$. However, this is still inefficient when the dataset is large or the trajectories are long. Our experiments show that it spends more than 30 minutes to answers a query on a dataset of 1 million trajectories, using NetEDR as distance function.

Observing the inefficiency of the above baseline methods, we resort to indexing trajectories offline and answering the online query with a *filter-and-verify* paradigm: by a filtering strategy, we first find a set of candidates, i.e., the data trajectories (along with the positions) that are probable to yield a match; and subsequently verify these candidates. Note that traditional filter-and-verify techniques for strings (e.g., q-grams and partition-based methods) are *inefficient* or *inapplicable* for WED on subtrajectories, as we have discussed in § 1.

3.1 Subsequence Filtering

The basic idea of our novel filtering principle, referred to as the *subsequence filtering*, is to choose a subsequence Q' of the query Q and derive a lower bound of WED using Q'. To guarantee to find all the answers to the query, we need a subsequence Q' such that if the lower bound reaches τ, then any trajectory P having a subtrajectory match to Q must contain at least one element (or one neighbor of the element) in Q'. To this end, we start with analyzing the effect of edit operations.

Given two (non-empty) trajectories P and Q on Σ, consider converting Q into P with some edit operations. If a symbol $q \in Q$ does not appear in P, q must be substituted or deleted from Q. For deletion, we need to pay a cost $\mathsf{sub}(q, \varepsilon)$. To substitute q with any $q' \in \Sigma \backslash \{q\}$, we need a cost $\mathsf{sub}(q, q')$. Therefore, to substitute or delete q, the cost is at least

$$\tilde{c}(q) := \min_{q' \in \Sigma^+ \backslash \{q\}} \mathsf{sub}(q, q').$$

Example 3 *Consider two trajectories $P = \texttt{BCD}$ and $Q = \texttt{ABC}$ on an alphabet $\Sigma = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$. Assume the following cost*

$\mathsf{sub}(\texttt{A}, \texttt{A})$	$\mathsf{sub}(\texttt{A}, \texttt{B})$	$\mathsf{sub}(\texttt{A}, \texttt{C})$	$\mathsf{sub}(\texttt{A}, \texttt{D})$	$\mathsf{del}(\texttt{A})$
0	5	3	6	4

Consider $q = \texttt{A}$ in Q. We see that $\texttt{A}$ does not appear in P and the minimum cost to delete or substitute this $\texttt{A}$ turns out to be $\tilde{c}(\texttt{A}) = \mathsf{sub}(\texttt{A}, \texttt{C}) = 3$.

Based on the cost $\tilde{c}(q)$, we can derive a lower bound. For the general case of WED, an issue is that this lower bound can become loose if there exists $q' \in \Sigma^+ \backslash \{q\}$ with a small cost; e.g., considering EDR, even if $q' \neq q$, $\mathsf{sub}(q, q')$ can be zero (when $d(q, q') \leq \varepsilon$). To address this issue and derive the filtering principle, we propose the concept of *substitution neighbors*.

Definition 4 (Substitution Neighbors) *Given a symbol $q \in \Sigma$, the* substitution neighbors *of q are defined by*

$$B(q) := \{b \in \Sigma \mid \mathsf{sub}(q, b) \leq \eta\}, \tag{5}$$

where $\eta \geq 0$ is a cost threshold, depending on the cost function (we discuss the choice of η below). For example, by setting $\eta = 0$ in EDR, $B(q)$ is a set of vertices $b \in V$ such that $d(q, b) \leq \varepsilon$ ($\Leftrightarrow \mathsf{sub}(q, b) = 0$). Note that $q \in B(q)$ always holds as $\mathsf{sub}(q, q) = 0$. Given a sequence $Q \in \Sigma^*$, we define the substitution neighbor of Q by

$$B(Q) := \bigcup_{q \in Q} B(q). \tag{6}$$

Intuitively, $B(Q)$ is comprised of the vertices (or edges) that are either in Q or too close to those in Q to deliver significant cost. By considering the costs of the elements in $B(Q)$, we

[1] The SW here is slightly different from the standard one which performs local alignment. We adapt SW to our problem by changing the boundary condition of dynamic programming. The pseudo-code is given in Appendix A of the extended version [21].

can obtain $c(Q)$, a lower bound of the cost of substituting or deleting all the elements in $B(Q)$, which leads to the following filtering principle [2].

Theorem 1 (Subsequence Filtering Principle) *Given a subtrajectory $P' \sqsubseteq P \in \Sigma^*$ and a query $Q \in \Sigma^*$, suppose a subsequence $Q' \subseteq Q$ such that $P' \cap B(Q') = \emptyset$ and $c(Q') := \sum_{q \in Q'} c(q) \geq \tau$ where*

$$c(q) := \min_{q' \in \Sigma^+ \setminus B(q)} \mathsf{sub}(q, q'). \tag{7}$$

Then $\mathsf{wed}(P', Q) \geq \tau$.

We refer to a subsequence Q' satisfying $c(Q') \geq \tau$ as a τ-*subsequence* of Q. The filtering principle states that if P' does not share any element with $B(Q')$, where Q' is a τ-subsequence of Q, then it is guaranteed that P' is not a result. Computing each $c(q), q \in Q'$is sublinear-time w.r.t. $|V|$ or $|E|$: For ERP, the complexity is $O(\log |V|)$ using a kd-tree. For other similarity functions in § 2.2, the complexity is $O(1)$, because $c(q)$ is 1 for EDR, Lev, and NetEDR, the smallest edge cost from q for NetERP, and $\mathsf{del}(q)$ for SURS. Further, we make two remarks on this filtering principle:

- The filtering is not limited to a specific cost function. Hence it can be used for the general purpose.
- Q' can be an arbitrary subsequence of Q (we discuss the choice of Q' in § 3.2).

Candidates are only found in the trajectories that pass the filtering principle. Since we are going to look up an inverted index with the elements in $B(Q')$ to identify candidates (§ 4), we denote each candidate by a triplet (id, j, i_q). id is the trajectory ID. j and i_q are positions in $P^{(id)}$ and Q, respectively, at which the candidate is identified; i.e., $P_j^{(id)} \in B(Q_{i_q}), Q_{i_q} \in Q'$.

Example 4 *Suppose $\eta = 0$ and the following cost matrix.*

q	$\mathsf{sub}(q, A)$	$\mathsf{sub}(q, B)$	$\mathsf{sub}(q, C)$	$\mathsf{sub}(q, D)$	$\mathsf{del}(q)$	$c(q)$
A	0	5	3	6	4	3
B	5	0	2	0	1	1
C	3	2	0	5	3	2
D	6	0	5	0	4	4

Consider $Q =$ ABC. By definition, we have $B(A) = \{A\}$, $B(B) = \{B, D\}$, $B(C) = \{C\}$, and $B(D) = \{B, D\}$. Taking the minimum over $\Sigma^+ \setminus B(q)$ for each row, we have $c(q)$ as in the table above. Consider $P^{(1)} =$ BCDBCD, $P^{(2)} =$ DABCBA, and $P^{(3)} =$ ABABAB. Consider $\tau = 3$ and $Q' = A \subseteq Q$, which satisfies $c(Q') \geq \tau$. As $P^{(1)} \cap Q' = \emptyset$, $P^{(1)}$ can be pruned (its subtrajectory closest to Q is BC, where $\mathsf{wed}(BC, Q) = \mathsf{del}(A) = 4 \not< \tau$). Since $P^{(2)}$ and $P^{(3)}$ contains A, they pass the filter and generate candidates: $(P^{(2)}, 2, 1)$, $(P^{(2)}, 6, 1)$, $(P^{(3)}, 1, 1)$, $(P^{(3)}, 3, 1)$, and $(P^{(3)}, 5, 1)$. By verification, only $(P^{(2)}, 2, 1)$ yields a result ABC because $\mathsf{wed}(ABC, Q) = 0 < \tau$. Other candidates are false positives (e.g., the closest subtrajectory to Q in $P^{(3)}$ is ABA, where $\mathsf{wed}(ABA, Q) = 3 \not< \tau$).

Choice of η. (1) For discrete cost functions (e.g., Lev and EDR), we can use $\eta = 0$, which excludes only symbols $a \in \Sigma^+$ with $sub(q, a) = 0$, $q \in Q'$. (2) For continuous cost functions (e.g., ERP), since the cost can be arbitrarily small, we need a small positive number for η to prevent the lower bound, $c(Q')$, becoming too loose (an extreme case is that $c(Q) < \tau$, making the choice of τ-subsequence impossible). We may tune η for the tightness of $c(Q)'$. With increasing η, $c(Q')$ increases, leading to a tighter lower bound; however, the number of symbols in $B(Q')$ also increases, which results in a larger candidate set. Setting η to $\frac{\tau}{|Q|}$ guarantees that a τ-subsequence can be found.

[2]The proofs are given in Appendix B of the extended version [21].

3.2 Finding Optimal τ-Subsequence

The subsequence filtering (Theorem 1) holds for any $Q' \subseteq Q$ that satisfies $c(Q') \geq \tau$. To reduce computational cost in verification, we propose to choose a subsequence that minimizes the number of candidates.

According to the subsequence filtering, $(P, T) \in \mathcal{T}$ such that $P \cap B(Q') \neq \emptyset$ will generate candidate trajectories. Let $n(q)$ be the frequency of a symbol $q \in \Sigma$ that appears in $\mathcal{T}$. We note that the frequency is counted multiple times if q occurs multiple times in a data trajectory, because we also record the positions j and i_q in a candidate. The total number of symbols in $\mathcal{T}$ that intersects with $B(q)$ is $\sum_{b \in B(q)} n(q)$. Therefore, we can formulate an optimization problem that minimizes the number of candidates as follows.

Definition 5 (Minimum Candidate Problem) *The minimum candidate problem* (MinCand) *is to find a subsequence $Q' \subseteq Q$ defined by the following discrete optimization problem:*

$$\min_{Q' \subseteq Q} \sum_{q \in Q'} \sum_{b \in B(q)} n(b), \quad \text{subject to} \quad \sum_{q \in Q'} c(q) \geq \tau. \tag{8}$$

Example 5 *Consider the same trajectories and cost matrix as Example 4. The frequencies are $n(A) = 5$, $n(B) = 7$, $n(C) = 3$, and $n(D) = 3$. There are seven subsequences of $Q =$ ABC, namely A, B, C, AB, AC, BC, and ABC. Among them, A, AB, AC, BC, and ABC satisfy the constraint $\sum_{q \in Q'} c(q) \geq \tau = 3$. Evaluating the objective function for each subsequence, we obtain:*

Q'	A	AB	AC	BC	ABC
Obj.	5	15	8	13	18

Hence, $Q' =$ A is the optimal solution (note: as $B(B) = \{B, D\}$, we have $\sum_{b \in B(B)} n(b) = n(B) + n(D) = 10$).

Remark. Given symbols q and q' in Q, if $B(q) \cap B(q') \neq \emptyset$, say q'' is a common element of $B(q)$ and $B(q')$, then Eq.(8) counts trajectories that travel on q'' twice. The elements counted multiple times are treated distinctly as they correspond to different candidates (distinct i_q and $i_{q'}$). To see the formulation does not violate the correctness of the search algorithm, there are $|Q'|$ elements in Q', each $q \in Q'$ has $|B(q)|$ substitution neighbors, and each neighbor b generates $n(b)$ candidates (i.e., the number of (id, j) pairs is $n(b)$ in $\mathcal{T}$). Besides, there is no duplicate among these candidates due to distinct (id, j) and i_q. Hence the objective in Eq.(8) is exactly the candidate size.

Next we discuss the computational aspects of MinCand. An observation is that MinCand is similar to the 0-1 knapsack problem. In fact, it can be reduced from the *Minimum Knapsack Problem* (MKP) [5], defined as follows.

$$\min_{S \subseteq [\![K]\!]} \sum_{k \in S} W_k, \quad \text{subject to} \quad \sum_{k \in S} V_k \geq D. \tag{9}$$

K is the number of items; W_k is the weight of an item and V_k is its value. The goal is to select a minimum weight subset of items $S \subseteq [\![K]\!]$, such that the total value is no less than a demand D. Hence we have

Proposition 2 MinCand *is NP-hard.*

Seeing the NP-hardness, we employ an approximation algorithm (Algorithm 1) based on [5] (MinCand can be reduced

Algorithm 1: MinCand(Q, n, c, τ)

```
1 N_q ← Σ_{b∈B(q)} n(b)   (∀q ∈ Q);          ▷ Trajectory freq.
2 Q' ← ∅;  w_q ← 0   (∀q ∈ Q);                ▷ Initialize
3 while τ > c(Q') do                          ▷ Constraint (8)
4     v_q ← (N_q − w_q)/ min{c(q), τ − c(Q')}   ∀q ∈ Q\Q';
5     q* ← argmin_{q∈Q\Q'} {v_q};             ▷ Choose greedily
6     w_q ← w_q + min{c(q), τ − c(Q')} · v_{q*}   ∀q ∈ Q\Q';
7     Q' ← Q' ∪ {(q*, i_{q*})};               ▷ i_{q*}:position of q* in Q
8 return Q'
```

to MKP and thus we can use the algorithm for MKP, see Proposition 3). In brief, this algorithm starts with an empty subsequence Q', and greedily adds an item q^* that has the minimum v_q value (Lines 4–5) to Q'. Intuitively, if $N_q/c(q)$ is small, the item would be worth choosing because we want to choose one with small N_q and large $c(q)$. Following the justification in [5], we extend this idea with a slight modification, and use $(N_q - w_q)/\min(c(q), \tau - c(Q'))$ as v_q, where the w_q variables (Line 6) are related to the dual problem of Eq. (9). We stop this procedure when the constraint in Eq. (8) (i.e., $\tau \leq c(Q') := \sum_{(q,i_q)\in Q'} c(q)$) is satisfied. At Line 7, we also record i_q, which is the position of q in Q. This information is carried when candidates are generated.

Example 6 *Suppose that* $Q =$ `ABCD`, $c = [1, 2, 3, 4]$ *and* $N = [5, 2, 9, 8]$. *If* $\tau = 4$, *we have* $v = [5, \underline{1}, 3, 2]$. *Hence, we add the second item,* `B`, *and its position, 2, to* Q'. *Then we update* $w = [1, 2, 3, 4]$ *and* $\tau - c(Q') = 4 - c(\texttt{B}) = 2$. *In the next iteration, we have* $v = [4/1, -, 6/2, 4/2]$ *and we add the forth item,* $(\texttt{D}, 4)$ *to* Q'. *This results in* $\tau - c(Q') = 4 - c(\texttt{B}) - c(\texttt{D}) = -2$ *and we stop the iteration and obtain* $Q' = \{(\texttt{B}, 2), (\texttt{D}, 4)\}$. *Although this* Q' *is not the optimal one* $Q^* = \{(\texttt{D}, 4)\}$, *we have a good approximation (10/8=25% loss compared to the optimal).*

Algorithm Property. Algorithm 1 runs in $O(|Q|^2)$ time. Further, the following statement holds.

Proposition 3 *Let* f^* *be the optimal objective value of Eq. (9). The approximation ratio of Algorithm 1 is 2, i.e., the approximated objective value is not greater than* $2f^*$.

For a special case, the following stronger result holds (EDR, Lev, and NetEDR satisfy this property).

Proposition 4 *If* $c(q)$ *is a constant function, i.e.,* $c(q) = c'$, *Algorithm 1 returns the optimal solution of* MinCand.

Solving MinCand is similar to finding best substrings (incl. q-grams) for string similarity problems [24, 25, 37, 48, 60]. The main differences are: (1) They mainly target Levenshtein distance on entire strings, while we cope with WED on substrings. (2) They resort to either heuristics [25, 37] or an offline constructed dictionary [24, 48, 60] without performance guarantee on candidate size, while we model this as a discrete optimization problem solved by a 2-approximation algorithm.

4. INDEXING AND SEARCH ALGORITHM

4.1 Indexing

Our indexing method employs inverted index [30], which is widely used for keyword search and also used to deal with trajectory similarity search (e.g., [47, 63]). We store data trajectories in the postings list (denoted by L_q) of each symbol $q \in \Sigma$. A record in L_q is in the form of (id, j), where id is the ID of a trajectory that passes q and j is its position, i.e., $P_j^{(id)} = q$. We can update the index by appending a new record to the corresponding postings list.

Algorithm 2: SubtrajSimSearch$(Q, \mathcal{T}, \mathsf{wed}, \tau)$

```
input : Query: Q; Database: T; Similarity function: wed;
        Similarity threshold: τ
1 Q' ← MinCand(Q, n, c, τ)        ▷ Optimize τ-subsequence
2 C ← ∅
3 for (q, i_q) ∈ Q' do
4     for b ∈ B(q) do
5         for (id, j) ∈ L_b do
6             C ← C ∪ {(id, j, i_q)}
7 A ← Verify(C, Q, τ)             ▷ See Algorithm 3 in § 5
8 return A
```

4.2 Search Algorithm

We propose an algorithm for subtrajectory similarity search problem based on the subsequence filtering in § 3. Algorithm 2 shows a skeleton of the algorithm.

Given a query Q, we first generate candidates based on Theorem 1. This states that for any subsequence Q' satisfying $c(Q') \geq \tau$, trajectories not included $\mathcal{C} = \cup_{b\in B(Q')} L_b$ can be safely pruned. To minimize the size of this candidate set $\mathcal{C}$, we solve the MinCand using Algorithm 1 (at Line 1 of Algorithm 2). We iterate through each $(q, i_q) \in Q'$ and look up the postings list of $b \in B(q)$. At Line 6, the ID of the data trajectory that contains $b \in B(q)$ is added to the candidate set. We also include the corresponding positions in P and Q (denoted by j and i_q, respectively). They are used to speed up the verification (§ 5). After the candidates are obtained, we verify whether each of them truly matches the query Q.

Incorporating spatial/road network indexing. Despite focusing on the general case of WED, it is noteworthy to mention that we can improve the query processing by indexing spatial information for a specific similarity function and regarding the index as a blackbox without needing to modify our algorithm. For similarity functions that involves Euclidean distance, we may index the coordinates of the vertices V using a spatial index, such as a kd-tree or an R-tree, so as to quickly compute $B(q)$ by retrieving the symbols within a range to q. For similarity functions involving shortest path distance (e.g., NetEDR and NetERP), we may use the *hub-labeling index* [1, 2] to compute shortest path distance to get $\mathsf{sub}(v, v')$. We provide a running example in the extended version [21].

4.3 Filtering with Temporal Information

As mentioned in § 2.3, we can treat any kind of temporal constraints as postprocessing. For interval constraints, such as $[T_i, T_j] \subseteq I$ or $[T_i, T_j] \cap I \neq \emptyset$, we can *prune candidates* before verification as follows. For each candidate trajectory $(P^{(id)}, T^{(id)})$ of length n, we check its first and last timestamps (i.e., $I^{(id)} := [T_1^{(id)}, T_n^{(id)}]$). Given a query time interval I, if $I^{(id)} \cap I = \emptyset$, then we have $[T_i, T_j] \not\subseteq I$ and $[T_i, T_j] \cap I \neq \emptyset$; we can safely prune this candidate. Furthermore, depending on the application, we may sort the records in each postings list by their temporal information such as departure time (i.e., $T_1^{(id)}$) or maximum speed (i.e., $\max_{1\leq t\leq |T^{(id)}|-1} w(P_t^{(id)})/(T_{t+1}^{(id)} - T_t^{(id)})$, where $w(e)$ is the distance of an edge e). This allows

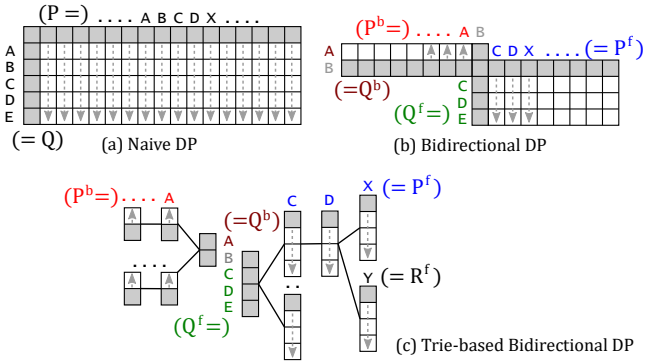

Figure 2: Comparison of verification methods (P = ABCDX..., Q = ABCDE; colors show partition).

us to *generate candidates* with binary search on postings lists, hence to avoid those violating the temporal constraint.

5. VERIFICATION

The generated candidates usually include many false positives; therefore, we need to verify them efficiently to obtain the answer. Existing methods for whole matching similarity search (e.g., [47, 63]) computes the similarity between P and Q with dynamic programming (DP) for each candidate. Similarly, in our subtrajectory search setting, we can compute the similarity by sequentially filling a $|P| \cdot |Q|$ matrix based on the recursive definition of WED, which is referred to as the Smith-Waterman (SW) algorithm. The time complexity is $O(|P| \cdot |Q|)$, as shown in Figure 2(a). This naive SW algorithm includes the following redundant computation: (1) Although a subtrajectory of P similar to Q can be a small part of P, the SW algorithm computes DP matrix for the entire P. (2) If two trajectories share a subtrajectory, the DP matrices have common values, but the SW algorithm does not exploit this property.

Main Idea. We reduce redundant computation as follows. (1) **Local verification**: Given a candidate (id, j, i_q), the subtrajectory similar to Q are located around the position j of P; hence we only need to run DP around j. (2) **Trie-based caching**: We share computation for common subtrajecories by exploiting the sparsity of a road network: Although the alphabet Σ is large, the possible previous/next symbols of P_j (i.e., P_{j-1} and P_{j+1}) are limited because trajectories move along physically connected vertices (edges) in a road network; hence we can efficiently cache the columns of DP matrices.

5.1 Local Verification

The filtering phase gives (id, j, i_q), where j is a position such that $P_j^{(id)}$ matches Q_{i_q} or its substitution neighbor. The local verification is to check if there exists a subtrajectory $P_{s:t}^{(id)}$ such that $\mathsf{wed}(P_{s:t}^{(id)}, Q) < \tau$, where $s \le j \le t$. Our idea is to run the DP computation from j *bidirectionally*. To guarantee that we will not miss any similarity search result after verifying all the candidates, we have the following lemma.

Lemma 1 *Given a subtrajectory $P_{s:t}^{(id)}$ of P such that $\mathsf{wed}(P_{s:t}^{(id)}, Q) < \tau$, and the set of candidates $\mathcal{C}$ identified by subsequence filtering, there exists $(id, j, i_q) \in \mathcal{C}$ such that $s \le j \le t$ and*

$$\mathsf{wed}(P_{s:t}^{(id)}, Q) = \mathsf{wed}(P_{s:j-1}^{(id)}, Q_{1:i_q-1}) + \mathsf{sub}(P_j^{(id)}, Q_{i_q}) + \mathsf{wed}(P_{j+1:t}^{(id)}, Q_{i_q+1:|Q|}). \quad (10)$$

This lemma suggests that for every subtrajectory that satisfies the WED constraint, we can always find a position j in the candidates identified by subsequence filtering, such that $P_j^{(id)}$ is aligned to Q_{i_q} in the optimal alignment that yields the WED. Thus, we can verify from j to obtain the similarity search result. Specifically, by Eq. (10), we partition $P_{s:t}^{(id)}$ into three parts at j and compute WED bidirectionally [15]. From j, we run two DPs: a **b**ackward one (i.e., from the end of strings to the start) for $\mathsf{wed}(P_{s:j-1}^{(id)}, Q_{1:i_q-1})$ and a **f**orward one for $\mathsf{wed}(P_{j+1:t}^{(id)}, Q_{i_q+1:|Q|})$. By iterating s from j to 1 and t from j to $|P^{(id)}|$ in the two DPs, respectively, we are able to find all $P_{s:t}^{(id)}$ such that $s \le j \le t$ and $\mathsf{wed}(P_{s:t}^{(id)}, Q) < \tau$. The above step is conducted for each (id, j, i_q) in the candidate set to obtain all the similarity search results. For ease of exposition, in the rest of this section, we omit the superscript (id) from P, and we use P^b to denote $P_{s:j-1}$ and P^f to denote $P_{j+1:t}$.

Example 7 *Consider a trajectory P = ...ABCDX... and a query Q = ABCDE. Assume a τ-subsequence is Q' = {B} and there is only one candidate whose $j = 2$ and $i_q = 2$. As shown in Figure 2(b), we partition P into (P^b, P_j, P^f) = (...A, B, CDX...) and Q into (Q^b, Q_{i_q}, Q^f) = (A, B, CDE). We can find all subtrajectories $P_{s:t}$ such that $s \le 2 \le t$ and $\mathsf{wed}(P_{s:t}, Q) < \tau$ by computing $\mathsf{sub}(B, B) + \mathsf{wed}(...A, A) + \mathsf{wed}(CDX..., CDE)$.*

Early Termination. For each direction $d \in \{b, f\}$, we can terminate the computation before reaching the end of P^d. Given a position $0 \le k \le |P^d|$ ($k = 0$ for the case of an empty string), by the definition of WED, we have the following lower bound of the WED between P^d and Q:

$$LB_k^d := \min_{0 \le j \le |Q|} \{\mathsf{wed}(P_{1:k}^d, Q_{1:j})\} \le \mathsf{wed}(P^d, Q). \quad (11)$$

If the lower bound for any k reaches τ, we can safely terminate the DP computation of P^d.

5.2 Caching with Bidirectional Trie

The local verification still involves redundant computation when we verify multiple candidates. We begin with an example.

Example 8 *(Continuing from Example 7) Consider another trajectory R = ...ABCDY... identified as a candidate via B. In the local verification, we need to compute both $\mathsf{wed}(P^f, Q^f)$ and $\mathsf{wed}(R^f, Q^f)$, where Q^f = CDE, P^f = CDX..., and R^f = CDY.... Hence, when computing $\mathsf{wed}(P^f, Q^f)$ and $\mathsf{wed}(R^f, Q^f)$, the first two columns of the DP matrices share the same values because P^f and R^f has a common prefix CD. This indicates that we can reduce the computation by caching these columns.*

In general, given a vertex/edge in the road network, the number of possible next vertices/edges are very small (typically, three) compared to the alphabet size because of the structure of the road network. This implies that the candidate subtrajectories starting from Q_{i_q} tend to share a prefix; therefore, we expect that the caching strategy will improve the efficiency.

To efficiently cache the DP columns of common prefixes, we employ a trie-based data structure as shown in Figure 2(c). We describe how this trie works using a running example.

Example 9 *(Continuing from Example 8) After computing $\mathsf{wed}(P^f, Q^f)$, we have three columns corresponding to C, D, and X, as in Figure 2(c). When computing $\mathsf{wed}(R^f, Q^f)$, we can reuse the two columns for C and D that are cached in the trie.*

Algorithm 3: Verify$(\mathcal{C}, Q, \tau)$

input : Candidates $\mathcal{C}$; Query Q; Threshold τ
1 **for** $(q, i_q) \in Q'$ **do**
2 $\mathcal{T}^f_{i_q}, \mathcal{T}^b_{i_q} \leftarrow$ Empty tries
3 **for** $(id, j, i_q) \in \mathcal{C}$ **do**
4 $\mathcal{A} \leftarrow \mathcal{A} \cup$ VerifyCandidate$(Q, (id, j, i_q), \tau, \mathcal{T}^f_{i_q}, \mathcal{T}^b_{i_q})$
5 **return** $\mathcal{A}$

Algorithm 4: VerifyCandidate$(Q, (id, j, i_q), \tau, \mathcal{T}^f, \mathcal{T}^b)$

input : Query: $Q \in \Sigma^*$; Candidate: (id, j, i_q); Threshold: τ; Forward/Backward tries: $\mathcal{T}^f, \mathcal{T}^b$
1 $P \leftarrow$ accessTrajectory(id)
2 $(P^b, b, P^f) \leftarrow (P_{1:(j-1)}, P_j, P_{(j+1):|P|})$ ▷ `Partition`
3 $(Q^b, q, Q^f) \leftarrow (Q_{1:(i_q-1)}, Q_{i_q}, Q_{(i_q+1):|Q|})$ ▷ `Partition`
4 $E^b \leftarrow$ AllPrefixWED$(Q^b, P^b, \tau', \mathcal{T}^b$.root$)$
5 $E^f \leftarrow$ AllPrefixWED$(Q^f, P^f, \tau', \mathcal{T}^f$.root$)$
6 **for** (s, t) **such that** $sub(q, b) + E^b_s + E^f_t < \tau$ **do**
7 $\mathcal{S} \leftarrow \mathcal{S} \cup \{(id, s, t)\}$ ▷ `Initialized as` $\mathcal{S} = \emptyset$
8 **return** $\mathcal{S}$ ▷ `All subtrajectories` $P_{s:t}$ `that matches` Q

As candidates tend to have common prefixes as discussed above, we expect that the cache miss rate is low and the verification gets faster. A trie is built for each direction (hence called a bidirectional trie) and each symbol in the τ-subsequence of Q. So there are $2|Q'|$ tries.

5.3 Verification Algorithm

Algorithms 3–6 summarize our verification algorithm. First, for each direction $\{b, f\}$ and $q \in Q'$ where Q' is a τ-subsequence of Q, empty tries are initialized (each trie is denoted by $\mathcal{T}^b_{i_q}$ or $\mathcal{T}^f_{i_q}$, where i_q indicates the candidate position in Q). Then, for each candidate $(id, j, i_q) \in \mathcal{C}$, VerifyCandidate (Algorithm 4) is applied and the results are stored in $\mathcal{A}$.

In VerifyCandidate, the data trajectory P and query Q are partitioned into three parts. Then, for each direction $d \in \{b, f\}$, AllPrefixWED is called to compute an array E^d, whose k-th element is the WED between Q^d and the k-th prefix of P^d, i.e., $E^d_k = \mathsf{wed}(P^d_{1:k}, Q^d)$. We can use a tighter threshold $\tau' := \tau - \mathsf{sub}(q, b)$ instead of the original τ because of Eq. (10); this τ' is used for early termination. Finally, all the subtrajectories that satisfy Eq. (10) $< \tau$ are added to the result set.

In AllPrefixWED (Algorithm 5), we compute the array E^d. A symbol c in a given trajectory P^d is processed one by one; if a child node corresponding to c at the current trie is found (Line 3), we skip computing the corresponding DP column; otherwise, we create a new child (Line 5) and compute the DP column (Line 6) using the StepDP procedure (Algorithm 6), a standard DP that computes a new column based on the previous column. $A^{(x)}$ represents a DP column cached in the trie node x. By Eq. (11), if the lower bound LB^d_k exceeds a given threshold (Line 7), we can safely terminate the DP computation for P^d. Finally, the value $A^{(x)}_{|Q_d|} = \mathsf{wed}(P^d_{1:k}, Q^d)$ is stored to E^d_k (Line 9).

Algorithm 5: AllPrefixWED(Q^d, P^d, τ, x)

input : Query Q^d; Trajectory P^d; Trie node x; $(d \in \{f, b\})$
output : WED between Q^d and $P^d_{1:k}$ $(\forall k)$
1 $E^d \leftarrow$ Empty array ▷ E^d_k `means` $\mathsf{wed}(Q^d, P^d_{1:k})$
2 **for** k **in** $1..|P^d|$ **do**
3 $c := P^d_k$; $x_{\mathsf{parent}} \leftarrow x$; $x \leftarrow x_{\mathsf{parent}}$.findChild$(c)$
4 **if** x not found **then**
5 $x \leftarrow x_{\mathsf{parent}}$.createChild$(c)$ ▷ `New child`
6 $A^{(x)} \leftarrow$ StepDP$(Q^d, c, A^{(x_{\mathsf{parent}})})$ ▷ `Fill DP column`
7 **if** $\tau \le \min_{0 \le j \le |Q|} A^{(x)}_j$ $(= LB^d_k)$ **then**
8 **break** ▷ `Early Termination (Sec.5.1)`
9 $E^d_k \leftarrow A^{(x)}_{|Q^d|}$ ▷ $\mathsf{wed}(Q^d, P^d_{1:k})$
10 **return** E^d

Algorithm 6: StepDP(Q^d, p, A)

input : Query Q^d; Next symbol $p \in \Sigma$; DP array $A_{0:|Q^d|}$
1 $B \leftarrow$ Array of length $|Q^d| + 1$; $B_0 \leftarrow A_0 + \mathsf{del}(p)$
2 **for** j **in** $1..|Q^d|$ **do**
3 $B_j \leftarrow \min\{A_{j-1} + \mathsf{sub}(p, Q^d_j), A_j + \mathsf{del}(p), B_{j-1} + \mathsf{ins}(Q^d_j)\}$
4 **return** B

Table 1: Dataset statistics.

Dataset	# Trajectories	Avg. Length	$\lvert V\rvert$	$\lvert E\rvert$
Beijing	786,801	101	86,484	171,135
Porto	1,701,238	81	75,265	135,133
SanFran	11,505,922	101	175,343	223,606

6. EXPERIMENTS

6.1 Settings

Due to the page limitation, we report the detailed experiment setup in [21], along with a set of additional experiments.

Evaluation was conducted on the following datasets: Beijing (T-drive) [64], Porto [32], and SanFran [4]. For Beijing and Porto, we conducted map matching [35] to obtain network-constrained representation. SanFran is a large synthesized dataset by the moving object generator [4] with the San Francisco road network. The statistics after preprocessing are presented in Table 1.

Our method consists of the (optimized) subsequence filtering with the bidirectional trie (BT) verification (referred to as OSF-BT). We also consider OSF-SW, where BT is replaced by the Smith-Waterman (SW) algorithm for verification. Further, we compare with the following baselines.

DISON. DISON [63] is a whole matching method for LCRS. We adapted it to our problem. Since the early termination technique in [63] does not work here, we used SW (DISON-SW) or BT verification (DISON-BT).

Torch. Torch [47] is a whole matching method that supports several similarity functions. We adapted it to our problem. We equipped it with SW (Torch-SW) and BT (Torch-BT) for verification. The upper bounding technique [47] prior to verification was developed for LORS and does not apply here.

DITA. DITA [41] is a whole matching method developed for DTW and can be adapted for other functions. We modified its pivoting method to fit WED. Since DITA does not support subtrajectory search, we enumerated all subtrajectories and indexed them. Note the enumeration is done offline and not counted towards query processing time. We used SW instead of the double-direction verification (DDV) [41] because DDV works for DTW but does not improve upon SW for WED.

q-gram indexing for EDR. q-gram indexing was proposed in [7] for whole matching under the EDR based on the fact that if there are less than $\max\{|P|, |Q|\} - q + 1 - \tau q$ common q-grams between P and Q, then $\mathsf{EDR}(P, Q) > \tau$. We customized their

method to support subtrajectory search under EDR. We set $q = 3$. SW was used for verification.

Indexing for ERP. ERP-index was proposed in [6]. It employs lower bounding and triangle inequality. Given a sequence P of coordinates, we indexed the sum of all coordinates, $sum(P) \in \mathbb{R}^2$, in a spatial index (we used kd-tree). Given a query sequence Q, $\|sum(P) - sum(Q)\|$ gives a lower bound. We enumerated and indexed all subtrajectories. SW was used for verification.

Smith-Waterman (SW). The SW algorithm [43] is a non-indexing method for substring matching. We adopted it (referred to as Plain-SW) to process all the data trajectories.

Since DITA and ERP-index enumerate all subtrajectories, the whole datasets are impossible to index due to exceeding the main memory (e.g., Beijing dataset generated 1.4 billion subtrajectories). We used a fraction of the dataset when these two methods were included in the competitors.

We used the six WED instances introduced in § 2.2 as similarity functions. Instead of specifying a similarity threshold τ directly, we used a threshold ratio $\tau_{\text{ratio}} \in [0, 1]$. Given a query Q and τ_{ratio}, we set $\tau := \tau_{\text{ratio}} \sum_{q \in Q} c(q)$. We used $\tau_{\text{ratio}} = 0.1$ as the default value. For SURS, LORS, and LCRS, costs are given by road lengths. The cost functions of the other WED instances are given in § 2.2. For EDR, we used $\varepsilon = 0.001$. For NetEDR, we set ε to the median distance of edges in E. For NetERP deletion cost $G^{(\text{del})}_{\text{NetERP}}$, we used $2M$. For η in Eq. (5), we used $\eta = 0$ for Lev, EDR, SURS, and NetEDR, 10^{-4} multiplied by the median distance of a node and its nearest neighbor for ERP, and the median road length for NetERP (see Appendix D of the extended version [21] for the choice of η).

All evaluations were conducted on a workstation with Intel Core i9-7900X CPU (3.30GHz) and 64GB RAM. All methods were implemented in C++ (g++ v.7.3.0) with the -O3 option. We do not use multi-threading (for distributed baselines, we implemented centralized versions to compare algorithms themselves). All the algorithms were implemented in a main memory fashion.

6.2 Effectiveness

6.2.1 Travel Time Estimation

To demonstrate the effectiveness of WED and subtrajectory similarity search, we first consider an on-the-fly travel time estimation task following the approach in [52]. We use the Beijing dataset and sample 130 queries of length 60, where the numbers of exact match are less than 10 (i.e., the sparse case). The travel times corresponding to the subtrajectories that exactly match the query Q are used as ground truth data. For estimation, we find the subtrajectories $\{P^{(id)}_{i:j}\}$ in the database similar to Q under a threshold τ_{ratio}. Since a trajectory $P^{(id)}$ may have multiple subtrajectories similar to Q, we pick the most similar one and break tie by the shortest one. Then we compute the average of the travel time, $T^{(id)}_j - T^{(id)}_i$, over those similar to Q as the estimated value. In order to evaluate the superiority of similarity search over exact match, we measure mean squared errors (MSEs) and report the relative value ($RMSE := MSE(\tau_{\text{ratio}})/MSE(\text{exact})$). Since the ground truths are contained in both results of similarity search and exact match, we employ a leave-one-out cross-validation by excluding one ground truth from the result set at a time. RMSE $< 100\%$ means similarity search is better than exact match.

We compare the six WED instances in § 2.2 with DTW, LORS, LCRS, and LCSS. Since LORS, LCRS, and LCSS are defined on shared road segments, we convert them to equivalent distance functions. DTW, LORS and LCSS are normalized to $[0, 1]$ such that $\text{DTW}(P, Q) \le \tau_{\text{ratio}} \sum_{i=1}^{|Q|-1} d(Q_i, Q_{i+1})^2$ and $\text{LORS}(P, Q) \ge (1 - \tau_{\text{ratio}}) \cdot \sum_{i=1}^{|Q|} w(Q_i)$ (the same for LCSS). Figure 3 shows how the RMSE changes over τ_{ratio} (LORS and LCSS have RMSE $> 100\%$ at $\tau_{\text{ratio}} = 0$ because they do not count mismatching road segments in $P^{(id)}$). For most WED instances, similarity search performs better than exact match when $\tau_{\text{ratio}} \in [0.04, 0.14]$, showcasing the superiority of similarity search for travel time estimation on sparse data. To compare similarity functions, all the WED instances except ERP perform well, while LORS and LCSS deliver the worst performance. SURS achieves the smallest RMSE (89%) among all the similarity functions. NetEDR and NetERP are competitive when τ_{ratio} is large. These observations suggest that WED is useful for travel time estimation and these new WED instances are better than existing ones (Lev, EDR, and ERP). We also observe that LCRS and SURS behave similarly because of similar semantics. An advantage of SURS over LCRS is that efficient subtrajectory search under LCRS has not been established and thus we enumerate all subtrajectories ($O(\sum_{k=1}^{N} |P|^2|Q|)$-time), while SURS belongs to WED and can be efficiently processed. Next, we compare similar subtrajectory matching with whole matching. Since whole matching finds no result for most thresholds, we consider a top-k setting for fair comparison. The RMSE is reported in Table 2 for SURS, the best function in the previous experiment. The result shows that the RMSE of subtrajectory matching is about half of the RMSE of whole matching, and the gap is more significant for small k.

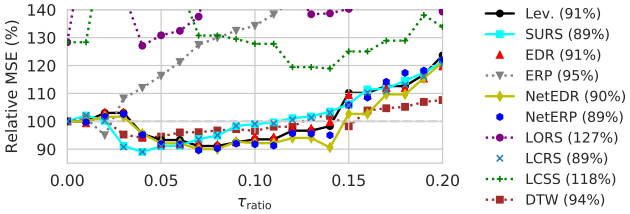

Figure 3: RMSE of travel time estimation (Beijing), best values reported on the right side.

Table 2: RMSE of travel time (SURS, Beijing).

k	5	10	15	20	25
Subtrajectory	92 %	91 %	102 %	108 %	116 %
Whole	233 %	221 %	219 %	220 %	220 %

6.2.2 Alternative Route Suggestion

Next we show the effectiveness through an alternative route suggestion task. Suppose a driver is planning to travel from an origin u to a destination v through a route Q, and the driver wants to find if there are variations of Q as alternative routes. We can do this by retrieving subtrajectories from u to v similar to Q from the database. To measure the preference of a route, we employ the route naturalness described in [65] §7: drivers prefer routes that go directly towards the destination, and the log-likelihood of a route is proportional to the number of hops that get closer (in terms of road network distance) to the destination than ever. Following this idea, given a route P such that $P_1 = u$ and $P_{|P|} = v$, we define its naturalness as the ratio of hops that get closer to v than ever, i.e., $\frac{|C|}{|P|-1}$, where $C = \{(P_{i-1}, P_i) \mid \min_{1 \le j < i} d(P_j, v) > d(P_i, v)\}$. If a route includes many inefficient detours, then the naturalness is low.

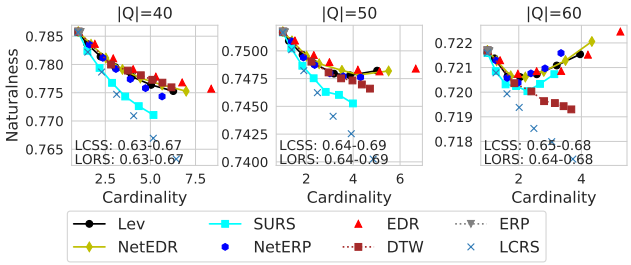

Figure 4: Naturalness of alternative routes suggested by similarity search $\tau_{\text{ratio}} \in [0, 0.3]$ (Beijing).

Table 3: Running time breakdown (ms).

	Default†	Varying τ_{ratio}		Varying $\|Q\|$	
		0.2	0.3	20	40
MinCand	0.002	0.005	0.007	0.0005	0.001
Index lookup	0.070	0.259	0.443	0.039	0.055
Verify	19.9	113.1	390.0	6.2	11.1

† Default: $\tau_{\text{ratio}} = 0.1$, $|Q| = 60$, dataset size = 100%.

Figure 4 shows the naturalness of the routes suggested by subtrajectory similarity search under various similarity functions. The results are averaged over 3,000 queries uniformly sampled from the Beijing dataset. We vary τ_{ratio} from 0 to 0.3 at 0.05 interval and plot the cardinality (i.e., the number of suggested routes) and the naturalness. The cardinality increases w.r.t. τ_{ratio}, but the rate depends on the similarity function. So we do not show τ_{ratio} explicitly. Among the six WED instances, Lev, EDR, NetEDR, and NetERP deliver routes with high naturalness. LCSS, LORS, and LCRS exhibit low naturalness because they measure common road segments but do not penalize inefficient detours. DTW's naturalness is high for short queries but drops rapidly for long queries. Another interesting observation is that when query length is 50 or 60, the naturalness using WED instances, except ERP, first decreases w.r.t. the cardinality and then rebounds. This is because some highly natural routes involve shortcuts that are spatially distant from the queries. They cause large DTW or ERP and hence are not identified using the two functions, while the WED instances with non-spatial distance as costs are capable of capturing these routes.

6.3 Query Processing Time

Following past related studies [20, 23, 50, 52], we randomly sampled subtrajectories from each dataset as queries. We set the default query length $|Q|$ to 60, whose path distance in real world ranges from 500 m to 40 km, with an average of 6.5 km, in line with [50, 52, 58]. Evaluation metrics were averaged over 100 queries, except for Plain-SW, whose processing time was averaged over 10 queries due to the computational cost.

We first investigate the effect of the similarity threshold τ_{ratio}. Figure 5 shows that the proposed method OSF-BT outperforms the other competitors. It responses in less than 2 seconds except for NetEDR, and typically hundreds of milliseconds when $\tau_{\text{ratio}} = 0.3$. Compared to DISON-BT and Torch-BT, which employ different filtering principles, our OSF-BT always performs better and is up to 9 times faster than DISON-BT and 73 times faster than Torch-BT. Comparing our BT verification with SW, BT significantly improves the efficiency. The impact of BT is more significant for NetEDR and NetERP, which involve relatively expensive computation in verification. For this reason, we omitted the results of DISON-SW and Torch-SW for NetEDR and NetERP from Figure 5, which take at least 24 hours for computation for 100 queries. These results indicate that both OSF and BT improve the performance, and improvements are consistently observed whatever similarity function is used.

We vary the length of query $|Q|$ with $\tau_{\text{ratio}} = 0.1$. As shown in Figure 6, OSF-BT is always faster than the others. For larger $|Q|$, the processing time of OSF-BT increases as well as the other methods. The reason is two-fold: (1) the verification cost is proportional to $|Q|$; (2) in our setting, the similarity threshold τ increases as $|Q|$ increases, making the candidate set larger.

In Table 3, we decompose the query processing time of OSF-BT (on Beijing, EDR) to: MinCand computation, index lookup, and verification. Most time (around 99%) is spent on verification, whose time increases with τ_{ratio} and $|Q|$. This is expected, because for each candidate we need only one index lookup but run a quadratic-time DP to verify it. MinCand computation is almost negligible since it runs in $O(|Q|^2)$ time, which does not depend on the dataset size.

Figure 7 shows query processing time when we vary the dataset size. All the methods scales linearly and OSF-BT is consistently the fastest. In addition, we show results for DITA and ERP-index, which requires subtrajectory enumeration, on a fraction of the datasets where they can fit into the main memory; e.g., in order to store the randomly chosen 5,000 trajectories, the number of subtrajectories to be indexed is 48M (Beijing), 37M (Porto), and 26M (SanFran). The query processing time for this small dataset is shown in Figure 8 (varying τ_{ratio}) and Figure 9 (varying dataset size). Our method outperforms DITA and ERP-index by two orders of magnitude. This result indicates that applying whole matching methods to subtrajectory matching by enumerating all subtrajectories is impractical for large datasets.

6.4 Filtering and Verification

We compare the filtering power by evaluating the candidate size $|\mathcal{C}|$. Note for all the competitors the candidates are in the form of (id, j, i_q) for fair comparison. Figure 10 shows: (1) OSF consistently results in the best filtering power; its candidate size is on average 3.4, 2.9, and 25 times smaller than DISON, q-gram, and Torch, respectively. (2) OSF shows good scalability against $|Q|$, because for long $|Q|$, the item set in MinCand becomes large and thus the probability of including "good-value-for-the-price" items becomes large. We also compare with DITA and ERP-index on a fraction of the datasets. Their candidate sizes are on average 105 and 14 times OSF's, respectively.

To separate the effect of local verification (§ 5.1) and the effect of caching with BT (§ 5.2), we evaluate (i) unpruned position rate (UPR), and (ii) cache miss rate (CMR), which are respectively defined as (i) the rate of DP columns that pass the early termination (§ 5.1) compared to SW, and (ii) the rate of DP columns where the StepDP procedure is actually called among the DP columns that pass the early termination. Table 4 shows results for the Beijing dataset under EDR. We observe both local verification and BT contribute to pruning. The rates increase with τ_{ratio} and $|Q|$ due to looser similarity constraint and longer query to verify, but decrease with dataset size due to more shared prefixes/suffixes. The total unpruned rate (TUR), defined by UPR × CMR, shows small values, indicating the number of StepDP calls is far less than that by SW.

6.5 Index Construction Time / Index Size

Table 5 the index construction time and index size. The index construction time of our postings lists is relatively fast. Among the competitors, only q-gram's index size is smaller than ours.

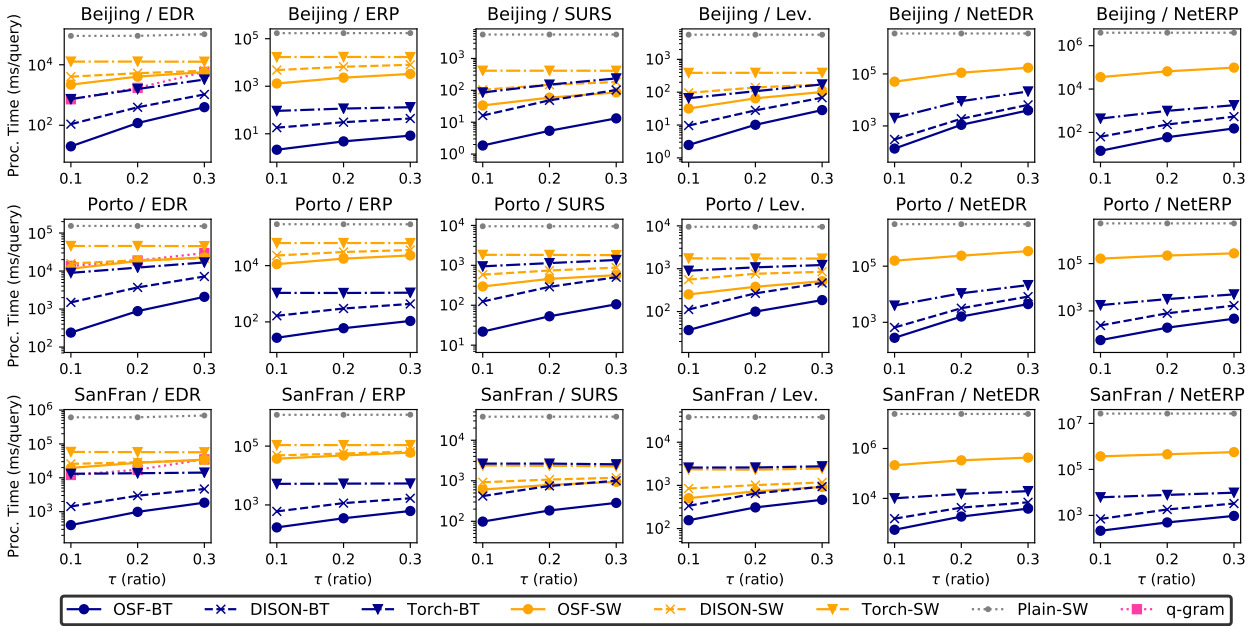

Figure 5: Varying τ_{ratio}: OSF-BT is our method (legend is shown at the bottom).

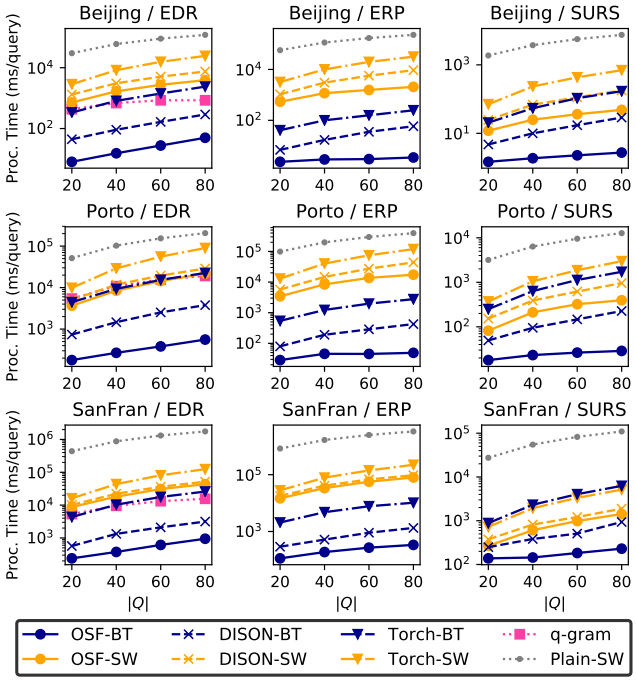

Figure 6: Varying $|Q|$.

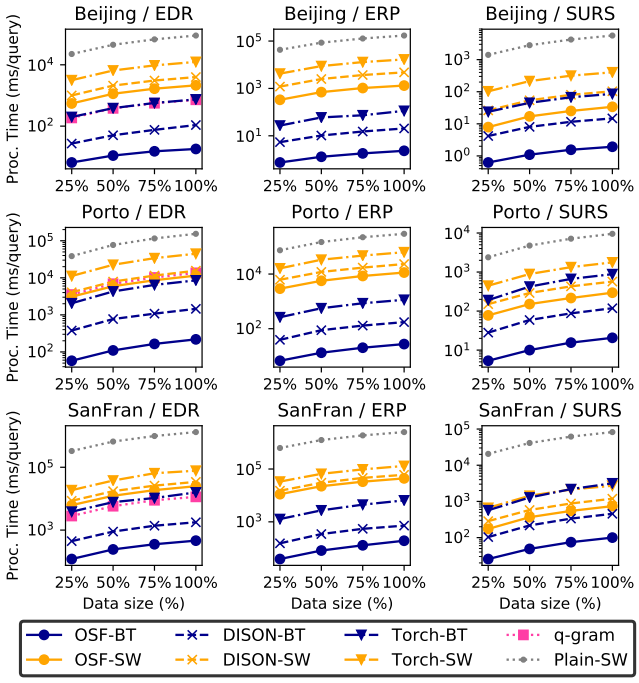

Figure 7: Varying $\mathcal{T}$.

Table 4: Evaluation of verification (%).

	Dflt.†	Varying τ_{ratio}		Varying $\|Q\|$		Varying $\|\mathcal{T}\|$	
		0.2	0.3	20	40	25%	50%
UPR	21.89	52.05	94.28	6.57	14.45	23.15	22.65
CMR	2.19	4.72	7.50	0.35	1.13	4.43	2.99
TUR	0.48	2.46	7.07	0.02	0.16	1.02	0.68

† Default: $\tau_{\text{ratio}} = 0.1$, $|Q| = 60$, dataset size = 100%.

For reference, we show results for methods involving subtrajectory enumeration (ERP-index and DITA). Although these are results with only 5,000 trajectories, the index construction time and index size are larger than ours except on SanFran.

7. RELATED WORK

Trajectory Similarity Functions. Trajectory similarity functions can be classified into: (1) coordinate-aware similarity functions are defined based on spatial coordinates of trajectories [6, 7, 42, 57, 62, 65] and (2) network-aware similarity functions employ network features, such as travel costs [11,14,39,40,45,47,54,55,63]. Coordinate-aware similarity func-

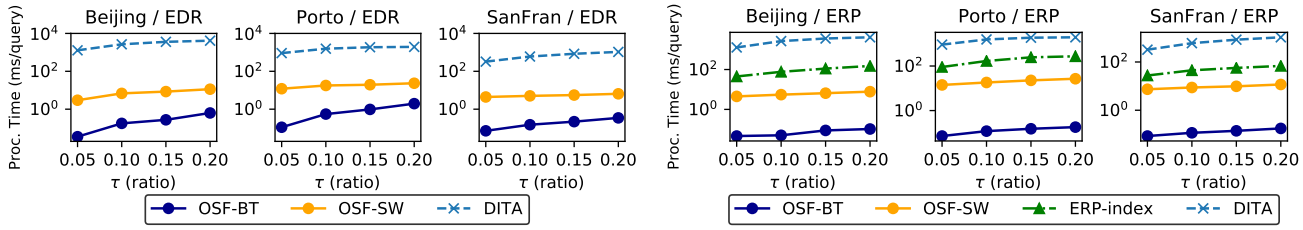

Figure 8: Comparison with baselines involving subtrajectory enumeration (Varying τ_{ratio}; $|\mathcal{T}| = 5000$; EDR/ERP).

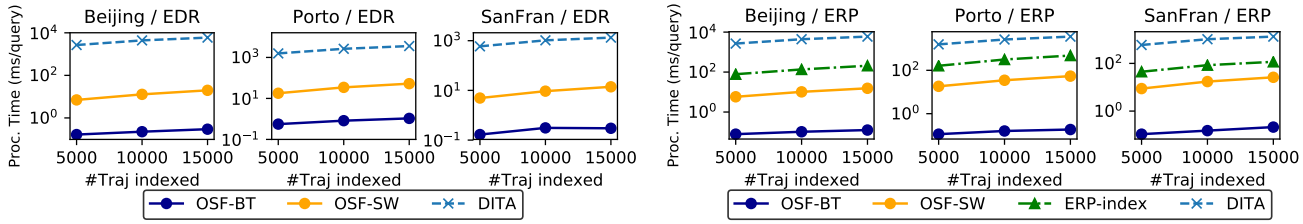

Figure 9: Comparison with baselines involving subtrajectory enumeration (Varying $|\mathcal{T}|$; $\tau_{\text{ratio}} = 0.1$; EDR/ERP).

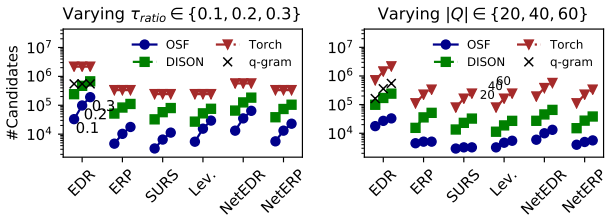

Figure 10: Number of candidate positions (Beijing).

Table 5: Index construction time / index size.

	Beijing	Porto	SanFran
OSF-BT†	7s / 0.59 GB	10s / 1.02 GB	79s / 8.63 GB
q-gram	15s / 0.59 GB	19s / 1.01 GB	269s / 8.55 GB
↓ Tiny dataset			
(ERP-index††)	23s / 2.6 GB	17s / 1.9 GB	14s / 1.4 GB
(DITA††)	60s / 1.79 GB	36s / 0.71 GB	31s / 0.15 GB

† DISON and Torch have the same time/size as OSF-BT.
†† Reference values with only 5,000 trajectories. DITA construction depends on the similarity function. We showed a result for ERP.

tions include dynamic time warping (DTW), edit distance with real penalty (ERP) [6], edit distance on real sequence (EDR) [7], edit distance with projections (EDwP) [38], and Fréchet distance. Their pros and cons were investigated in [6, 7, 44, 47, 63]. ERP and EDR are WED instances. DTW, EDwP, and Fréchet distance are not. Recent effort aimed to learn deep trajectory representations [28] or metrics [61] to reduce the computation of similarity to linear time. For network-aware similarity functions, a natural way is to measure the shared or unshared edges. Weighted Jaccard distance [55] and weighted Dice distance [54] are order-insensitive functions (i.e., edge ordering not incorporated). Longest common subsequences (LCSS) [33, 46], Longest overlapping road segments (LORS) [47], and longest common road segments (LCRS) [63] are order-sensitive functions, while they do not belong to WED. Another strategy is to incorporate shortest path distances between vertices [11, 14, 39, 40, 45].

Trajectory Indexing. Although much attention has been gathered to indexing for non-constrained trajectories (see surveys [31, 36]), indexing methods for trajectories in road networks are also studied actively [9, 20, 22, 23, 47, 63]. Chen et al. proposed ERP [6] and EDR [7] along with the query processing algorithms. Wang et al. [47] and Yuan and Li [63] proposed algorithms to support various similarity functions but they were designed for whole matching (though can be adapted for subtrajectories, see our experiments). Furthermore, their filtering policies are different from ours (based on scanning postings lists for all symbols [47] or prefix symbols [63] of a query). Shang et al. [41] and Xie et al. [57] proposed distributed systems for similarity search under DTW and discrete Fréche/Hausdorff distances, respectively. These methods [41, 57] support only whole matching. Pivot points were proposed in [41] for pruning. The differences from our method are: (1) The pivots are selected by turning points or the distance to neighbor points/origin/destination, while our τ-subsequence is chosen by a selectivity optimization algorithm. (2) In contrast to pivot points, our filtering is designed towards the general case of WED, and thus does not require individual adaptation for specific similarity functions.

String / Time-series Similarity Search. Edit distance has been employed for string similarity search. Bouding techniques (e.g., by q-grams [10, 37, 53]) are widely used. In bioinformatics, three types of similarity search methods are used: global [34], local [27, 43, 59], and semi-global alignments [26]. The Smith-Waterman algorithm [43] can be used for WED. Trajectories also belong to time series data, whose similarity is often measured by Euclidean distance, DTW, ERP, or LCSS. Common indexing methods are based on lower bounding [3, 17].

8. CONCLUSION AND FUTURE WORK

We tackled subtrajectory similarity search under WED, a class of similarity function that includes several important similarity functions. For efficient search, we proposed subsequence filtering, which involves a discrete optimization problem to choose the optimal subsequence. Based on this technique, we developed an algorithm using filter-and-verify strategy. We designed a local verification method equipped with bidirectional tries. We showed the effectiveness of WED and the superiority of our solution over alternative methods. Interesting future directions include supporting more general class of similarity functions, developing a distributed indexing method, and more sophisticated treatment of temporal information.

Acknowledgments. This work was supported by JSPS 16H01722, 17H06099, 18H04093, 19K11979, and NSFC 61702409.

9. REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *ESA*, pages 24–35, 2012.

[2] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, pages 349–360, 2013.

[3] I. Assent, R. Krieger, F. Afschari, and T. Seidl. The TS-tree: Efficient Time Series Search and Retrieval. In *EDBT*, pages 252–263, 2008.

[4] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[5] T. Carnes and D. B. Shmoys. Primal-dual schema for capacitated covering problems. *Mathematical Programming*, 153(2):289–308, 2015.

[6] L. Chen and R. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[7] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[8] Z. Chen, H. T. Shen, and X. Zhou. Discovering popular routes from trajectories. In *ICDE*, pages 900–911, 2011.

[9] V. T. de Almeida and R. H. Güting. Indexing the trajectories of moving objects in networks. *Geoinformatica*, 9(1):33–60, 2005.

[10] D. Deng, G. Li, and J. Feng. A pivotal prefix based filtering algorithm for string similarity search. In *SIGMOD*, pages 673–684, 2014.

[11] M. R. Evans, D. Oliver, S. Shekhar, and F. Harvey. Fast and exact network trajectory similarity computation: a case-study on bicycle corridor planning. In *UrbComp@KDD*, pages 9:1–9:8, 2013.

[12] J. Feng, J. Wang, and G. Li. Trie-join: a trie-based method for efficient string similarity joins. *VLDB J.*, 21(4):437–461, 2012.

[13] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[14] J.-R. Hwang, H.-Y. Kang, and K.-J. Li. Searching for similar trajectories on road networks using spatio-temporal similarity. In *ADBIS*, pages 282–295, 2006.

[15] H. Hyyrö and G. Navarro. A practical index for genome searching. In *SPIRE*, pages 341–349, 2003.

[16] T. Idé and S. Kato. Travel-time prediction using gaussian process regression: A trajectory-based approach. In *SDM*, pages 1185–1196, 2009.

[17] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[18] S. Koide, K. Kawano, and T. Kutsuna. Neural edit operations for biological sequences. In *Advances in Neural Information Processing Systems 31*, pages 4965–4975, 2018.

[19] S. Koide, Y. Tadokoro, C. Xiao, and Y. Ishikawa. CiNCT: Compression and retrieval for massive vehicular trajectories via relative movement labeling. In *ICDE*, pages 1097–1108, 2018.

[20] S. Koide, Y. Tadokoro, T. Yoshimura, C. Xiao, and Y. Ishikawa. Enhanced indexing and querying of trajectories in road networks via string algorithms. *ACM Trans. Spatial Algorithms Syst.*, 4(1):3:1–3:41, 2018.

[21] S. Koide, C. Xiao, and Y. Ishikawa. Fast subtrajectory similarity search under weighted edit distance constraints. *CoRR*, abs/2006.05564, 2020.

[22] B. Krogh, C. S. Jensen, and K. Torp. Efficient in-memory indexing of network-constrained trajectories. In *GIS*, pages 17:1–17:10, 2016.

[23] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp. Path-based queries on trajectory data. In *GIS*, pages 341–350, 2014.

[24] C. Li, B. Wang, and X. Yang. VGRAM: improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, pages 303–314, 2007.

[25] G. Li, D. Deng, and J. Feng. A partition-based method for string similarity joins with edit-distance constraints. *ACM Trans. Database Syst.*, 38(2):9:1–9:33, 2013.

[26] H. Li and R. Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[27] H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[28] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei. Deep representation learning for trajectory similarity computation. In *ICDE*, pages 617–628, 2018.

[29] W. Luo, H. Tan, L. Chen, and L. M. Ni. Finding time period-based most frequent path in big trajectory data. In *SIGMOD*, pages 713–724, 2013.

[30] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[31] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-Temporal Access Methods. *IEEE Data Eng. Bull.*, 26(2):40–49, 2003.

[32] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas. Predicting taxi-passenger demand using streaming data. *IEEE Trans. Intelligent Transportation Systems*, 14(3):1393–1402, 2013.

[33] M. D. Morse and J. M. Patel. An efficient and accurate method for evaluating time series similarity. In *SIGMOD*, pages 569–580, 2007.

[34] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 3 1970.

[35] P. Newson and J. Krumm. Hidden Markov map matching through noise and sparseness. In *GIS*, pages 336–343, 2009.

[36] L.-V. Nguyen-Dinh, W. G. Aref, and M. F. Mokbel. Spatio-Temporal Access Methods : Part 2 (2003 – 2010). *IEEE Data Eng. Bull.*, 33(2):46–55, 2010.

[37] J. Qin, W. Wang, C. Xiao, Y. Lu, X. Lin, and H. Wang. Asymmetric signature schemes for efficient exact edit similarity query processing. *ACM Trans. Database Syst.*, 38(3):16:1–16:44, 2013.

[38] S. Ranu, D. P, A. D. Telang, P. Deshpande, and S. Raghavan. Indexing and matching trajectories under inconsistent sampling rates. In *ICDE*, pages 999–1010, 2015.

[39] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Trajectory Similarity Join in Spatial Networks. *PVLDB*, 10(11):1178–1189, 2017.

[40] 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.

[41] Z. Shang, G. Li, and Z. Bao. DITA: Distributed In-Memory Trajectory Analytics. In *SIGMOD*, pages 725–740, 2018.

[42] C.-B. Shim and J.-W. Chang. Similar sub-trajectory retrieval for moving objects in spatio-temporal databases. In *ADBIS*, pages 308–322, 2003.

[43] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[44] H. Su, S. Liu, B. Zheng, X. Zhou, and K. Zheng. A survey of trajectory distance measures and performance evaluation. *VLDB J.*, 29(1):3–32, 2020.

[45] E. Tiakas, A. Papadopoulos, A. Nanopoulos, Y. Manolopoulos, D. Stojanovic, and S. Djordjevic-Kajan. Searching for similar trajectories in spatial networks. *Journal of Systems and Software*, 82(5):772 – 788, 2009.

[46] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[47] S. Wang, Z. Bao, J. S. Culpepper, Z. Xie, Q. Liu, and X. Qin. Torch: A search engine for trajectory data. In *SIGIR*, pages 535–544, 2018.

[48] W. Wang, J. Qin, C. Xiao, X. Lin, and H. T. Shen. Vchunkjoin: An efficient algorithm for edit similarity joins. *IEEE Trans. Knowl. Data Eng.*, 25(8):1916–1929, 2013.

[49] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient approximate entity extraction with edit distance constraints. In *SIGMOD*, pages 759–770, 2009.

[50] Y. Wang, Y. Zheng, and Y. Xue. Travel time estimation of a path using sparse trajectories. In *KDD*, pages 25–34, 2014.

[51] R. Waury, J. Hu, B. Yang, and C. S. Jensen. Assessing the accuracy benefits of on-the-fly trajectory selection in fine-grained travel-time estimation. In *MDM*, pages 240–245, 2017.

[52] R. Waury, C. S. Jensen, S. Koide, Y. Ishikawa, and C. Xiao. Indexing trajectories for travel-time histogram retrieval. In *EDBT*, pages 157–168, 2019.

[53] H. Wei, J. X. Yu, and C. Lu. String similarity search: A hash-based approach. *IEEE Trans. Knowl. Data Eng.*, 30(1):170–184, 2018.

[54] J. I. Won, S. W. Kim, J. H. Baek, and J. Lee. Trajectory clustering in road network environment. In *CIDM*, pages 299–305, 2009.

[55] Y. Xia, G. Y. Wang, X. Zhang, G. B. Kim, and H. Y. Bae. Spatio-temporal Similarity Measure for Network Constrained Trajectory Data. *Int. J. Comput. Intell. Syst.*, 4(5):1070–1079, 2011.

[56] C. Xiao, W. Wang, and X. Lin. Ed-Join : An Efficient Algorithm for Similarity Joins With Edit Distance Constraints. *PVLDB*, 1(1):933–944, 2008.

[57] D. Xie, F. Li, and J. M. Phillips. Distributed trajectory similarity search. *PVLDB*, 10(11):1478–1489, 2017.

[58] L. Yang, Y. Wang, Q. Bai, and S. Han. Urban Form and Travel Patterns by Commuters: Comparative Case Study of Wuhan and Xi'an, China. *Journal of urban planning and development*, 144(1):05017014, 2018.

[59] X. Yang, H. Liu, and B. Wang. ALAE: Accelerating Local Alignment with Affine Gap Exactly in Biosequence Databases. *PVLDB*, 5(11):1507–1518, 2012.

[60] X. Yang, B. Wang, and C. Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *SIGMOD*, pages 353–364, 2008.

[61] D. Yao, G. Cong, C. Zhang, and J. Bi. Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach. In *ICDE*, pages 1358–1369, 2019.

[62] B. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1998.

[63] H. Yuan and G. Li. Distributed in-memory trajectory similarity search and join on road network. In *ICDE*, pages 1262–1273, 2019.

[64] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *GIS*, pages 99–108, 2010.

[65] Y. Zheng and X. Zhou. *Computing with Spatial Trajectories*. Springer, 2011.