Finding Long and Similar Parts of Trajectories

Kevin Buchin

Maike Buchin

Marc van Kreveld

Jun Luo

Technical Report UU-CS-2011-012
May 2011

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Finding Long and Similar Parts of Trajectories *

Kevin Buchin Maike Buchin
Department of Mathematics and Computer Science
Technical University Eindhoven
The Netherlands
{k.a.buchin, m.e.buchin}@tue.nl

Marc van Kreveld
Department of Information and Computing Sciences
Utrecht University
The Netherlands
marc@cs.uu.nl

Jun Luo
Shenzhen Institute of Advanced Technology
Chinese Academy of Sciences, China
jun.luo@sub.siat.ac.cn

Abstract

A natural time-dependent similarity measure for two trajectories is their average distance at corresponding times. We give algorithms for computing the most similar subtrajectories under this measure, assuming the two trajectories are given as two polygonal, possibly self-intersecting lines with time stamps. For the case when a minimum duration of the subtrajectories is specified and the subtrajectories must start at corresponding times, we give a linear-time algorithm. The algorithm is based on a result of independent interest: We present a linear-time algorithm to find, for a piece-wise monotone function, an interval of at least a given length that has minimum average value. In the case that the subtrajectories may start at non-corresponding times, it appears difficult to give exact algorithms, even if the duration of the subtrajectories is fixed. For this case, we give $(1+\varepsilon)$-approximation algorithms, for both fixed duration and when only a minimum duration is specified.

1 Introduction

Trajectory analysis With the widespread usage of mobile location-aware devices, huge amounts of trajectory data are captured every day. It becomes more and more interesting to analyze the trajectories of moving objects such as people, animals, vehicles, and natural phenomena, e.g., hurricanes. A key operation for analyzing trajectories is determining their similarity. This can be used for *prediction*, *clustering*, and further analysis tasks. Prediction of trajectories is often based on similar older trajectories, which are retrieved from a database.

Clustering, i.e., grouping objects by similarity, is an important task in spatio-temporal data mining. For trajectories an important analysis task is detecting *movement patterns*. Patterns that have been considered include flocks, convoys, commuting, migration, and many others [12].

*A preliminary version of this paper appeared at ACM GIS '09. This research has been partially supported by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant no. 642.065.503 and project no. 639.022.707 and by the German Research Foundation (DFG) under grant no. BU 2419/1-1.

If we ignore the time component of a trajectory, it is simply a polygonal line that may self-intersect and have duplicate vertices. Similarity and distance measures for polygonal lines have been developed many years ago, motivated by line simplification and shape matching. Such measures include the areal displacement, the Hausdorff distance, and the Fréchet distance. Since these measures are shape-dependent only and do not consider temporal aspects such as speed, they are not suitable for trajectory analysis. For example, when analyzing hurricane trajectories, we must distinguish between hurricanes that go fast and those that go slow, even if they follow exactly the same route. Figure 1 shows two trajectories τ_1 and τ_2 that have the same shape and a similar location, but the speed development of the two trajectories is very different: trajectory τ_1 is fast at first, and then slows down, whereas this is reversed for trajectory τ_2. We require from a similarity measure that it reflects the difference of these two trajectories.

As remarked, an important analysis type for a collection of trajectories is clustering [16, 19]. Figure 2 shows a set of six trajectories whose clusters we may want to determine. Two clusters are easily visible, but it is not directly clear if the dashed trajectory, τ_3, fits better with $\{\tau_1, \tau_2\}$ or with $\{\tau_4, \tau_5, \tau_6\}$. Based on shape and distance similarity, it fits slightly better in the first group, but if we consider time-dependent similarity like the average distance at corresponding times (defined formally later), it fits better in the second group, basically because the speeds are more similar.

Trajectories can also be seen as time series data or sequences of locations, and can thus be handled by techniques from time series analysis and sequence comparison. In particular, several trajectory similarity measures based on such techniques have been developed. In contrast to the measure considered in this paper, these measures do not directly use the Euclidean distance or do not require corresponding times.

Subtrajectory Similarity Often the similarity of parts of two trajectories is more important than the similarity of the whole trajectories. This was observed before by Lee et al. [17], who motivate that subtrajectory similarity may be more appropriate for clustering purposes than similarity of complete trajectories. For instance, we may want to know whether two entities were traveling together for some time, but not necessarily for the whole route. Also, the starting time of a trajectory may be somewhat arbitrary (a hurricane builds up speed, it does not start out of nowhere), or its initial behavior may be atypical for the subsequent trajectory. Tourists may take very similar walking routes through a city, but they will start at different hotels. Similarly, hurricanes may start to follow certain routes only later in their lifetime. Such similarity information can be important for the prediction of the subsequent route, and we therefore require a similarity measure on subtrajectories instead of whole trajectories. To make sure that the similarity occurs during a sufficiently long period, we will require a minimum duration.

Furthermore, we want to be able to find similar subtrajectories that may begin at different starting times in the input trajectories. For this, we allow a *shift* to obtain the proper time correspondence. In Figure 3 we need to allow a *time shift* to be able to find the similarity of the middle part of τ_1 and the last part of τ_2; assuming the locations are recorded every half hour, the time shift would be roughly 2 years, 2 days, 2 hours and 45 minutes.

Related research This paper presents algorithms to find the most similar subtrajectories of two trajectories with or without a time shift, using the average distance over the subtrajectories at corresponding times. This similarity measure has been considered previously [9, 15, 19, 20], however, the results in this paper extend and improve all of these previous approaches. Sinha and

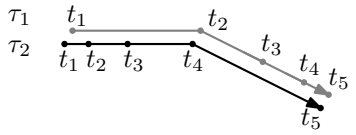


Figure 1: Trajectories with similar shape but different speeds at corresponding times.

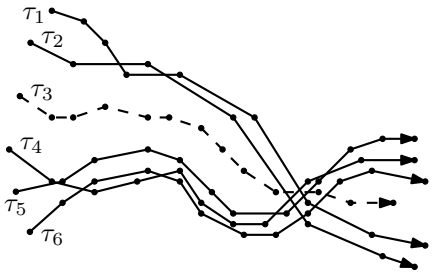


Figure 2: Clustering six trajectories in two groups.

Mark [20] give algorithms only for discrete trajectories. Nanni and Pedreschi [19] give algorithms for the continuous case, but they assume that the trajectories start at the same moment in time, and they do not deal with subtrajectory similarity. Van Kreveld and Luo [15] give algorithms for the continuous case, allowing for subtrajectory similarity and time shifts. In this paper, we substantially improve their algorithms, from quadratic to linear-time for subtrajectory similarity without time shift and from a heuristic to a guaranteed approximation for subtrajectory similarity with time shift. Frentzos et al. [9] also recently considered this similarity measure. However, instead of computing it exactly, they approximate the integral using the trapezoid rule.

Next we discuss related trajectory similarity measures. Trajcevski et al. [21] use the maximum instead of the average distance at corresponding times as similarity measure. They give algorithms for optimal matching under rotations and translations. They do not consider time shifts and subtrajectory similarity.

A related dissimilarity measure, inspired by time series analysis, is considered by Yanagiswa et al. [24]. They use the Euclidean distance of the trajectories as n-dimensional vectors. That is, at each time stamp they use the squared Euclidean distance. By squaring the distances, the effect of outliers is strengthened, which is typically not desirable for trajectories, in particular if they contain noise.

Several further methods from time series analysis have been applied or adapted for computing trajectory similarity. Dynamic time warping (DTW) [14] has been used for trajectory similarity. Furthermore, trajectory similarity measures have been developed based on longest common subsequences (LCSS) [22], dynamic time warping (DTW) [23], and edit distance [8]. These measures all differ in nature to the average distance at corresponding times. DTW, edit distance, and LCSS do not require to match points at corresponding times, but allow a more flexible matching. The trajectory similarity measure based on LCSS [22] introduces a bound on the time shift. Instead of directly using Euclidean distance between points, in this measure [22] as well as the measure based on edit distance [8], distances are quantized to 0 or 1 by thresholding the Euclidean distance. DTW can be used with Euclidean distance between points [14] or difference in turning angle [23].

A geometric distance measure to find similar subtrajectories is considered by Buchin et al. [4]. They give algorithms for subtrajectory similarity with time shift under the Fréchet distance, which does not require an exact time correspondence. In fact, in the discrete case, this measure coincides with DTW under the Euclidean distance. A different solution to partial similarity under the Fréchet distance is given by Buchin et al. [5]. The Fréchet distance as similarity measure for trajectories has been further developed by Buchin et al. [3]. They show how to incorporate additional

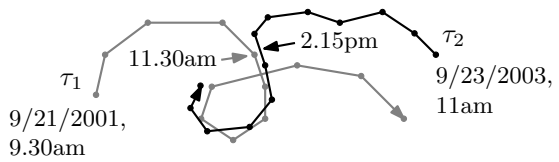


Figure 3: The middle part of trajectory τ_1 is similar to the last part of trajectory τ_2.

constraints, for instance on the time correspondence, direction, or speed of the trajectories.

Lee et al. [17] have given a method for subtrajectory clustering using similarity of line segments derived from the trajectories. The similarity measure used for the line segments is geometric only, and does not take the temporal component into account. Hence, their clustering method may be more appropriate for polygonal shapes than for trajectories.

Problem statement Let $\tau_1(.), \tau_2(.)$ be parameterizations of two trajectories over time. The dissimilarity, or distance, between the part of τ_1 starting at time t_s and the part of τ_2 starting at time $t_s + t_{shift}$, both with the same duration $T > 0$, is defined as [15]:

$$\frac{\int_{t_s}^{t_s+T} d(\,\tau_1(t)\,,\,\tau_2(t+t_{shift})\,)\,dt}{T}$$

where $d(.,.)$ is the Euclidean distance between $\tau_1(.)$ and $\tau_2(.)$. This is the average distance at corresponding times, where the time correspondence is determined by t_{shift}. We are interested in the combination of T, t_s, and t_{shift} that gives the smallest dissimilarity.

There are four variants of the subtrajectory similarity problem using the average distance at corresponding times as dissimilarity measure. The duration of the time interval, T, may be specified beforehand, or it may be of any length bounded from below by a minimum $T_{\min}$. Furthermore, the starting times of the subtrajectories may be specified to be the same or not (depending on the application). If they are the same, then $t_{shift} = 0$ and it can be omitted from the dissimilarity measure. If the starting times need not be the same, we will say that we allow a time shift, and t_{shift} is one of the unknowns over which we optimize.

Results of this paper Without time shift and with a fixed duration, a simple algorithm finds the most similar subtrajectories of τ_1 and τ_2 in $O(n)$ time, where n is the number of vertices in each of the trajectories. Only t_s is unknown in this version of the problem. Without time shift but with a non-fixed duration, this extends to an $O(n^2)$ time algorithm [15]. Here, both t_s and T are unknown, and $T \geq T_{\min}$ for a given value of $T_{\min}$ is required. In this paper we improve the quadratic-time algorithm substantially by presenting an optimal, linear-time algorithm.

For subtrajectory similarity with an unknown time shift, an algorithm that computes the exact solution to the most similar subtrajectory problem is unlikely to exist, and in [15] heuristic algorithms were therefore given. These algorithms did not have any quality guarantees. In this paper we show that quality guarantees on the approximation are possible, and give $(1 + \varepsilon)$-approximation algorithms for both the fixed-duration and non-fixed duration case. This means that if the optimal solution has a dissimilarity of D, then our algorithm will find a solution where the dissimilarity is no more than $(1 + \varepsilon) \cdot D$. Here, ε is a fixed positive constant that must be specified beforehand. We can for instance specify that the average distance of the approximate solution is at most 5% higher than the best possible average distance. The running time of our algorithms is $O(n^4/\varepsilon)$ if the duration of the subtrajectories is fixed, and $O(n^4/\varepsilon^2)$ if the duration is not fixed, but bounded from below. If the observation times $t_1, t_2, t_3, \ldots$ are regularly spaced on each of the trajectories, then our time bounds are improved by a linear factor to $O(n^3/\varepsilon)$ and $O(n^3/\varepsilon^2)$, respectively.

Overview In the next section we give some observations on the most similar subtrajectory and approximations of it. In Section 3 we give a linear-time algorithm for subtrajectory similarity of non-fixed duration without time shift. It is based on a more general geometric algorithm that computes an interval of minimum average value of a piecewise monotone function of a least some length. In Section 4 we present and analyze $(1 + \varepsilon)$-approximation algorithms for subtrajectory similarity with time shift.

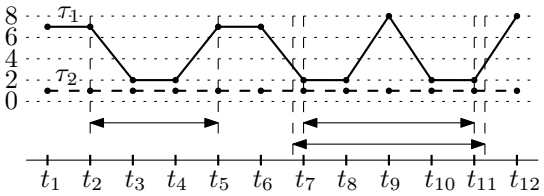


Figure 4: Subtrajectory similarity with non-fixed duration.

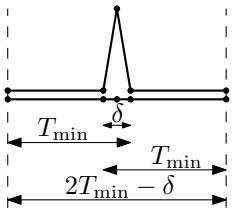


Figure 5: Worst-case bound between fixed and non-fixed duration.

2 Preliminary observations

In our problem statements, we require a minimum duration $T_{\min}$ of the two subtrajectories. If we would not have this requirement, the most similar subtrajectories would always have duration 0: the single moment where the two trajectories were closest. This suggests that shorter trajectories have a larger similarity than longer trajectories. It appears that this intuition is only true in part, which we explain next.

Figure 4 shows that allowing a longer duration than $T_{\min}$ can give more similar subtrajectories. Assume that no time shift is allowed. If the duration T is fixed to be three units, then $[t_2, t_5]$ gives the lowest average distance value, namely $8/3$. If the duration may be longer, then $[t_7, t_{11}]$ gives the lower average distance of $5/2$. Extending the interval even more to start just before t_7 and end just after t_{11} gives an even lower average distance value. The example also shows that the optimal time interval need not have its endpoints at vertices of the trajectories.

For a fixed minimum duration $T_{\min}$, the highest similarity is always attained by a pair subtrajectories of duration between $T_{\min}$ and $2T_{\min}$. This can be seen as follows. Any pair of subtrajectories of longer duration can be split into two pairs of half the duration (which are still at least $T_{\min}$ long). The pair with the lower dissimilarity must have a lower dissimilarity than the pair of full subtrajectories (although it may be the same). Repeating the argument will lead to a pair of subtrajectories of duration between $T_{\min}$ and $2T_{\min}$ whose dissimilarity is at most that of a pair with duration larger than $2T_{\min}$.

We also observe that if the interval with lowest dissimilarity has duration T', with $T_{\min} \leq T' \leq 2T_{\min}$, then the dissimilarity for any subinterval of length $T_{\min}$ is at most by a factor $T'/T_{\min}$ larger. Thus, the dissimilarity for fixed duration $T_{\min}$ is at most a factor 2 larger than for non-fixed duration with lower bound $T_{\min}$. The factor 2 can be realized in the limit, as demonstrated in Figure 5: The total duration is $2T_{\min} - \delta$ and the distance between the trajectories is 0 except for a duration of δ in the middle (for illustration purposes the trajectories are shown with a small vertical offset). In this example, fixed and non-fixed duration similarity differ by a factor of $(2T_{\min} - \delta)/T_{\min}$ which is $2 - \delta$ for $T_{\min} = 1$. If we let δ approach 0, the factor 2 is realized in the limit.

3 Exact algorithm for subtrajectories without time shift

In this section we consider computing the most similar subtrajectories without a time shift. If the duration is fixed, a simple linear-time algorithm is to sweep a "time window" over the distance

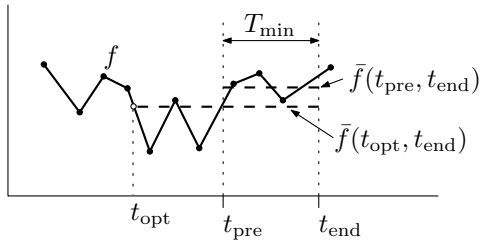


Figure 6: Averages of f over $[t_{\text{pre}}, t_{\text{end}}]$ and $[t_{\text{opt}}, t_{\text{end}}]$.

function and maintain its average. For non-fixed duration, we also sweep over the distance function, but now we maintain a data structure storing more information about swept intervals. For two single line segments of the input trajectories, the distance function is hyperbolic in t. Over the whole trajectories, it is piecewise hyperbolic. On each piece it has no local maxima and possibly one local minimum interior to the interval. By splitting intervals at such minima, the distance function is a piecewise monotone function.

In fact, we will solve a more abstract version of the problem: Given a piecewise monotone function f and a value $T_{\min}$, find a subinterval I of the domain of f of length at least $T_{\min}$, such that the average value of f over I is minimized. For this problem, we give an algorithm with a running time linear in the number of monotone pieces of f.

The discrete version of this problem occurs in biological sequences alignment, where one usually wants to maximize: Given a sequence of real numbers and a lower bound L, find a consecutive subsequence of length at least L that has maximum average. This problem is called the *maximum density segment problem*. Several linear-time algorithms have been given for this problem [2, 7, 10, 18]. Conceptually our algorithm is similar to the algorithm by Bernholt et al. [2], since both are based on convex hulls. But the convex hulls computed are completely different. Our algorithm uses a sweep similar to the algorithms in [7, 10, 18], however, our non-discrete version is more general and considerably more complex, due to allowing any start and end point of the interval and any piecewise monotone function.

3.1 Concept

We first illustrate the idea of the algorithm. We solve the problem by a sweep over the domain of f. At any time t_{end} we include at least a window of the minimum length $T_{\min}$ to the left of t_{end}. Additionally it may lower the average to include a part even further to the left. To decide efficiently how much of this part to include, we decompose and store this part in a data structure.

Let t_{end} be the end of some time interval; we are interested in a minimum average value of f over an interval of length at least $T_{\min}$ that ends at t_{end}. Let $t_{\text{pre}} = t_{\text{end}} - T_{\min}$ be the last moment where the interval can start. But we may want to start the interval earlier, to lower the average value of f over the chosen interval (see Figure 6). We need a careful analysis of the situation before t_{pre} to decide what the optimal starting time is for an interval that ends at t_{end}. We will store this situation in a data structure that will be updated when t_{end} and t_{pre} move simultaneously further in time. There will be events when t_{end} or t_{pre} pass a break point of the function f, but there will also be events if the situation before t_{pre} changes in a structural way.

For any interval $I = [t', t'']$ we define $\bar{f}(t', t'')$ as

$$\bar{f}(I) := \bar{f}(t', t'') = \frac{\int_{t'}^{t''} f(t)dt}{t'' - t'} .$$

If $t' = t''$, we let $\bar{f}(t', t'') = \bar{f}(t', t') := f(t')$. We also define $\bar{f}(t') := \bar{f}(t', t_{\text{end}})$, which is well-defined if t_{end} is fixed.

For description purposes, we fix t_{end} and therefore t_{pre} for the moment. Then interval $[t_{\text{pre}}, t_{\text{end}}]$

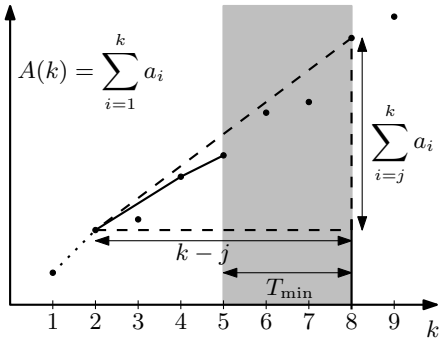


Figure 7: Geometric interpretation of the algorithm.

gives an average of

$$\bar{f}(t_{\text{pre}}) = \bar{f}(t_{\text{pre}}, t_{\text{end}}) = \frac{\int_{t_{\text{pre}}}^{t_{\text{end}}} f(t)dt}{t_{\text{end}} - t_{\text{pre}}} .$$

It is clear that if the function value of f is smaller than $\bar{f}(t_{\text{pre}})$ just before t_{pre}, then extending the interval to a starting time before t_{pre} will give a lower average $\bar{f}(.)$. But even if the function value of f just before t_{pre} is greater than $\bar{f}(t_{\text{pre}})$, then extending the interval to a starting time (sufficiently far) before t_{pre} may still give a lower average. In Figure 6 we observe that the optimal starting time $t_{\text{opt}} \leq t_{\text{pre}}$, given t_{end} as the end, is such that $\bar{f}(t_{\text{opt}}) = f(t_{\text{opt}})$, or $t_{\text{opt}} = t_{\text{pre}}$.

If t_{end} is fixed, then only the *value* of $\bar{f}(t_{\text{pre}})$ determines where t_{opt} is. The time t_{opt} is monotonically decreasing in the value of $\bar{f}(t_{\text{pre}})$: if the average of f over $[t_{\text{pre}}, t_{\text{end}}]$ were larger, we may go further back with t_{opt}, but never forward.

3.2 Geometric interpretation

Although we give the algorithm using a non-geometric representation, conceptually it can be seen as a convex hull construction. Consider the graph of the integral $F(t) := \int_{t_1}^{t} f(s)ds$. The average $\bar{f}(t', t'')$ corresponds to the slope of the secant from $(t', F(t'))$ to $(t'', F(t''))$. Thus, we are looking for the secant with the smallest slope. In this context, the algorithm presented here corresponds to a convex hull construction by a sweep.

Figure 7 illustrates the geometric interpretation in the discrete setting, i.e., for a sequence. Assume we have a sequence $a_1, \ldots, a_n$. The discrete analogue of the integral $F(t)$ is the partial sum $A(k) := \sum_{i=1}^{k} a_i$. We are looking for the pair of numbers j, k between 1 and n with $k \geq j + T_{\min}$ minimizing $\frac{A(k)-A(j)}{k-j}$, thus, minimizing the slope of the line through $(j, A(j))$ and $(k, A(k))$. In the figure, we consider $k = 8$ and are looking for the optimal $j \leq 5$. The properties we will use (in the continuous setting) are:

- The point $(j, A(j))$ must be on the convex hull of $(1, A(1)) \ldots (k - T_{\min}, A(k - T_{\min}))$, otherwise the point on the convex hull left to it would have been better. In Figure 7, for instance, $j = 2$ is a better choice than $j = 3$.

- For a given k, if j is optimal for k then the line through $(j, A(j))$ and $(k, A(k))$ is tangent to the convex hull, i.e., the line does not cross the interior of the convex hull. Otherwise there would be a point above the line which is better. For instance, if we choose any $j \neq 2$ in the figure, we can decrease the slope of the line by rotating the line around $(8, A(8))$ until it goes through $(2, A(2))$.

- If a point j_0 was optimal for a k_0 then for all $k > k_0$ we do not have to consider points to the left of j_0. The reason for this is that any line that is tangent to a vertex of the convex hull left to j_0 would have a larger slope. For instance, in the figure any line that is tangent to the convex hull at $(1, A(1))$ would have a larger slope than the line through $(2, A(2))$ and

$(8, A(8))$. As a consequence, during the sweep the optimal left endpoint only moves to the right (or remains where it is). Thus, we only need to maintain the convex hull to the right of the current value j.

In the following we will describe the algorithm directly in terms of $\bar{f}(t', t'')$.

3.3 Assumptions on f

In our algorithm we assume that the following operations can be performed in constant time:

1. Evaluate the integral of f over a monotone piece.
2. Solve equations of the form $F(a, s) = bs + c$, where $F(a, s) = \int_a^s f(t)dt$ and a, s lie on the same monotone piece of f.
3. Find an interval of minimum average value, if the monotone pieces for the left and the right endpoint of the interval are given and the integral of f for the intervals in between has been evaluated.

These assumptions are reasonable for the distance function of two trajectories. For the first and second assumption this is obvious. We discuss the third assumption below. For simplicity, we will also assume that f is continuous. We can extend our method to handle non-continuous functions, but the description of the algorithm becomes more cumbersome.

In general, how difficult is it to fulfill the third assumption? Assume the union of the intervals in between is $[t_0, t_0']$, and let $D := t_0' - t_0$. We assume that the integral of f has been already evaluated over $[t_0, t_0']$, thus, we know the average value C of f over this interval. Let f_1 and f_2 correspond to f on the monotone piece to the left and the monotone piece to the right, more specifically, $f_1(t) := f(t_0 - t)$ and $f_2(t) := f(t - t_0')$. We need to minimize

$$\frac{\int_0^{t_1} f_1(t)dt + C + \int_0^{t_2} f_2(t)dt}{t_1 + D + t_2}.$$

In a minimum the partial derivatives are zero. Transforming this gives

$$f_1(t_1) = f_2(t_2) = \frac{\int_0^{t_1} f_1(t)dt + C + \int_0^{t_2} f_2(t)dt}{t_1 + D + t_2}.$$

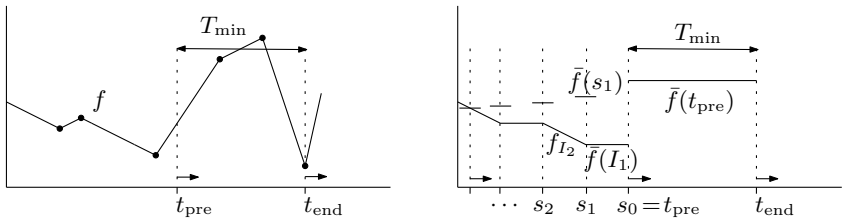
We can now transform $f_1(t_1) = f_2(t_2)$ to $t_2 = s(t_1)$ and then solve

$$f_1(t_1) = \frac{\int_0^{t_1} f_1(t)dt + C + \int_0^{s(t_1)} f_2(t)dt}{t_1 + D + s(t_1)}.$$

If f_1 and f_2 are linear functions, then the above equation is quadratic. If f_1 and f_2 are hyperbolic, as in our application, we get closed-form expressions for the integrals resulting in a constant-size equation. Thus it is reasonable to assume that we can minimize the distance function for two trajectories in fixed start and end intervals.

3.4 Data structure

Let f be a piecewise monotone function with break points $t_1, \ldots, t_n$, i.e., f is monotone in between each pair t_i and t_{i+1} for $1 \leq i < n$. At all times our data structure consists of the interval $I_0 = [t_{\text{pre}}, t_{\text{end}}]$ and a set of intervals $I_1, \ldots, I_{m(t_{\text{pre}})}$, where $I_i = [s_i, s_{i-1}]$, for $i = 1, \ldots, m(t_{\text{pre}})$, $m(t_{\text{pre}}) \geq 0$, and $s_{m(t_{\text{pre}})} < s_{m(t_{\text{pre}})-1} < \ldots < s_1 < s_0 = t_{\text{pre}}$, cf. Figure 8. To avoid cluttered notation we let $m := m(t_{\text{pre}})$, but we keep in mind that the number of intervals depends on t_{pre}, and changes throughout the algorithm. To define $s_1, \ldots, s_m$ and m, we first define a function $l(s)$



(a) distance function (b) data structure

Figure 8: Data structure maintained during the sweep for non-fixed duration.

which, intuitively, tells us how far to the left we can always extend an interval if we extend it at least a fraction to the left of s, and still lower the average $\bar{f}$. We define l on the domain of f by

$$l(s) := \min(s' \leq s \mid \forall\, 0 \leq t \leq s : \bar{f}(s', s) \leq \bar{f}(t, s)\ )$$

Note that if for no $s' < s$ we have $\bar{f}(s', s) < f(s)$, then $l(s) = s$. This can only happen if f is a decreasing function at s (on its left side).

We can now define the interval endpoints s_i, $1 \leq i \leq m$, by

$$s_i := \begin{cases} l(s_{i-1}) & \text{if } l(s_{i-1}) < s_{i-1} \\ \max(\,\{t_j < s_{i-1} \mid 1 \leq j \leq n\,\} & \\ \qquad \cup \{s' < s_{i-1} \mid l(s') < s'\,\}\,) & \text{else.} \end{cases}$$

Thus, if $l(s_{i-1}) = s_{i-1}$, then we set s_i either to the next break point t_j of f left of s_{i-1}, or to the largest $s' < s_{i-1}$ such that $l(s') < s'$. If $s_i = l(s_{i-1})$, then f must be decreasing just left of s_i.

There are two types of intervals in $I_1, \ldots, I_m$: those where $s_i = l(s_{i-1})$ and those where $s_i < l(s_{i-1})$. We will call the first type of intervals *complete* and the other type *decreasing*. These intervals have the following properties:

1. If I_i is complete and $i > 1$, then $f(s_{i-1}) = f(s_i) = \bar{f}(I_i)$.
2. If I_i is complete, then for all $s' \in (s_i, s_{i-1})$ we have $\bar{f}(s_i) < \bar{f}(s')$.
3. If I_i is complete and $i \geq 1$, then I_{i+1} is decreasing.
4. $\bar{f}(I_i) < \bar{f}(I_j)$ if and only if $i < j$.

Note that the first property does not hold for I_1 because it is not preceded by a decreasing interval. The last property states that the average gets higher to the left. Any complete interval contains a break point of f, and consecutive decreasing intervals are separated by a break point of f. Together with the third property, this implies $m = O(n)$.

The integer m, representing the last interval to the left that we need to consider, depends on the average value of f over intervals in the data structure. We do not need to consider intervals at the left end of our data structure if their (partial) inclusion would increase the average. It follows that the last interval I_m is a decreasing interval. Also, we do not need intervals further to the left if their inclusion would result in an average value that is larger than a previously found average value. Hence, we have:

$$f(s_m) \;\geq\; \min_{t' + T_{\min} \leq t \leq t_{\text{end}}} \bar{f}(t', t) \;\geq\; f(s_{m-1})\ .$$

Our data structure maintains the sequence of break points $t_{\text{end}}, s_0, s_1, \ldots, s_m$ and for each, the piece of f that contains it, and the sequence $F(I_0), \ldots, F(I_m)$, where $F(I_i) = \int_{s_{i-1}}^{s_i} f(t)dt$. The sequences can simply be stored in a list or an array. During the algorithm, we only change information at the ends of the sequences. We also maintain $F(I_{m-1} \cup \cdots \cup I_1)$.

3.5 Algorithm

To find the interval with minimum average value, we scan with the interval $[t_{\text{pre}}, t_{\text{end}}]$ from start to end along the domain of f, maintaining the information just described. Most of this information only changes at certain discrete event points that we handle during the scan. The positions of t_{end}, s_0, and possibly s_1 change continuously, but we will use the maintained information and their notation as it was valid at the last event. We use t'_{end}, s'_0, s'_1, I'_0, etc., to denote the corresponding values that are valid at the next event, and $\tilde{t} = \tilde{t}_{\text{end}}$, $\tilde{t}_{\text{pre}} = \tilde{s}_0$, $\tilde{s}_1$, $\tilde{I}_0$, etc., to denote the values in between events t_{end} and t'_{end}.

In between two consecutive event points $t_{\text{end}} \leq \tilde{t} \leq t'_{\text{end}}$, we need to minimize $\bar{f}(t, \tilde{t})$ over the choices of t and $\tilde{t}$ with $t \leq \tilde{t} - T_{\min}$, knowing on which pieces of f the interval endpoints t and $\tilde{t}$ lie. To minimize $\bar{f}(t, \tilde{t})$, we find the expressions for $F(t, \tilde{s}_{m-1}) = F(t, s_{m-1})$ and $F(\tilde{I}_{m-1} \cup \cdots \cup \tilde{I}_0) = F(I_{m-1} \cup \cdots \cup I_3 \cup \tilde{I}_2 \cup \tilde{I}_1 \cup \tilde{I}_0)$ in the unknowns t and $\tilde{t}$, and minimize. Since $t \in I_m$, and I_m is a decreasing interval, we have one piece of f over I_m. Hence, the expression for $F(t, s_{m-1})$ is easy to obtain in constant time. Also, we maintained $F(I_{m-1} \cup \cdots \cup I_1)$ and all $F(I_i)$ at t_{end}, and $\tilde{t}$ does not pass any vertex of f before the next event. So we can determine the expression $F(\tilde{I}_{m-1} \cup \cdots \cup \tilde{I}_0)$ in constant time as well (if $m = 0$, we simply take the expression $\bar{f}(\tilde{t} - T_{\min}, \tilde{t})$). By the third assumption, we can minimize such expressions in constant time. Summarizing, we can find the optimal interval between two consecutive events in constant time.

It remains to describe how we update the data structure in constant time. Instead of precomputing all event points, we will compute them dynamically.

3.5.1 Event points

Recall that t_{end} denotes the time of the previous event and t'_{end} denotes the time of the next event. We have four types of events.

1. *(break)* $\tilde{I}_0$ moves to the next break point of f, that is, either $s'_0 = t_i$ or $t'_{\text{end}} = t_i$ for $1 \leq i \leq n$. If $s'_0 = t_i$ and I_1 is decreasing, then we create a new interval I'_1 starting at s'_0, which may be decreasing or complete.

2. *(merge)* I_1 is complete, and $\bar{f}(\tilde{I}_1)$ increases until $\tilde{I}_2$ disappears. If I_3 is decreasing, then $I'_1 = [s_2, s'_0]$; otherwise, $I'_1 = [s_3, s'_0]$ (two adjacent complete intervals merge).

3. *(create)* I_1 is complete, $\bar{f}(\tilde{I}_1)$ increases and $f(\tilde{s}_0)$ decreases until $\bar{f}(\tilde{I}_1) = f(\tilde{s}_0)$. We create a new decreasing interval I'_1 starting at s'_0.

4. *(discard)* The leftmost interval becomes irrelevant because its average value is too large, that is, $\bar{f}(s_{m-1}) \geq \min_{t \leq t'_{\text{end}} - T_{\min}} \bar{f}(t, t'_{\text{end}})$. Then we discard I_m. If I_{m-1} is complete, we discard it as well.

Note that instead of stopping at an event of type 4, we can also check if it has happened at the next event of type 1, 2, or 3, and process it first.

3.5.2 Computing the event points

The event points of type 1 are the break points of f, and they are known beforehand. We cannot precompute the event points of types 2, 3, and 4, but we can compute the next such event point if it is before the next type 1 event. We do this as follows. Let t_{end} be the most recent event point, and let $s_0, s_1, \ldots$ be the interval endpoints with respect to t_{end}. Let t'_{end} be the next event point of type 1. An event point $\tilde{t}$ of type 2 occurs for $t_{\text{end}} < \tilde{t} < t'_{\text{end}}$ if $f(s_2) = \bar{f}(s_2, \tilde{t} - T_{\min})$. To detect this, we observe

$$\bar{f}(s_2, \tilde{t} - T_{\min}) = \frac{F(s_2, s_1) + F(s_1, s_0) + F(s_0, \tilde{t} - T_{\min})}{\tilde{t} - T_{\min} - s_2}$$

and make an expression in $\tilde{t}$. This takes constant time using the values $F(I_2)$, $F(I_1)$, and $F(I_0)$. Then we find $\tilde{t}$ by setting it equal to $f(s_2)$, which is possible by the second assumption. Similarly,

we detect an event of type 3 by solving $f(\tilde{t} - T_{\min}) = \bar{f}(s_2, \tilde{t} - T_{\min})$. An event point $\tilde{t}$ of type 4 occurs for $t_{\text{end}} < \tilde{t} < t'_{\text{end}}$ if $\bar{f}(s_{m-1}, \tilde{t}) = f(s_{m-1})$. To detect this we solve

$$\bar{f}(s_{m-1}, \tilde{t}) = \frac{F(s_{m-1}, t_{\text{end}}) + F(t_{\text{end}}, \tilde{t})}{\tilde{t} - s_{m-1}} = f(s_{m-1}),$$

which we can compute in constant time as well.

3.5.3 Updating the data structure

At all types of event points we update the interval endpoints t_{end}, s_0, and s_1 in constant time. At an event of type 2 we discard s_1 and possibly s_2. At an event of type 4 we discard I_m and s_m, and if I_{m-1} is complete, we discard it and s_{m-1} as well. In all cases we update $F(I_0)$, $F(I_1)$, $F(I_2)$, and $F(I_{m-1} \cup \cdots \cup I_1)$, and the pieces of f that contain t_{end}, s_0, s_1, and s_m. Each of these updates takes only constant time.

3.5.4 Correctness

The optimal solution is the minimal value $\bar{f}(t, \tilde{t})$ for $t \leq \tilde{t} - T_{\min}$. Assume $[t, \tilde{t}\,]$ is an optimal interval with the following properties: (i) $\tilde{t}$ is minimal if t is given, and (ii) t is maximal if $\tilde{t}$ is given. Let $t_{\text{end}} < \tilde{t} < t'_{\text{end}}$ be the event points just before and after of $\tilde{t}$. We need to prove that $t \in I_m$.

Suppose $t < s_m$. Then t lies in a previously discarded interval. Let t'' be the right endpoint of this interval before it was discarded. Since the interval was discarded, there are $\hat{t}_{\text{start}}, \hat{t}_{\text{end}}$ with $\hat{t}_{\text{start}} \leq \hat{t}_{\text{end}} - T_{\min}$ and $\hat{t}_{\text{end}} \leq \tilde{t}$ such that $\bar{f}(t, t'') \geq \bar{f}(\hat{t}_{\text{start}}, \hat{t}_{\text{end}})$. Because of optimality of $[t, \tilde{t}\,]$, $\bar{f}(t, t'') \geq \bar{f}(t, \tilde{t})$. Therefore, discarding the interval from t to t'' will not increase the average. This contradicts the maximality of t.

Next, suppose $s_i < t \leq s_{i-1}$ for some $i \leq m - 1$. But then adding the interval from s_i to t to the interval $[t, \tilde{t}\,]$ would decrease the average because I_m was not discarded, which contradicts the optimality of $[t, \tilde{t}\,]$.

3.5.5 Run-time

The run-time is linear in the number of events because the cost per event point is constant. The number of type 1 event points is $2n$. At an event of type 2, two or three intervals are merged into one. At an event of type 3, one decreasing interval is created as I'_1, but this can only happen if I_1 was complete. At an event of type 4 one or two intervals are removed. After a type 3 event, we cannot have a type 2 or 3 event next. Hence, the number of event points of type 2, 3, and 4 is bounded by the total number of intervals created and is therefore at most linear in n.

Theorem 1. *Given $T_{\min} > 0$ and a piecewise monotone function f consisting of n monotone pieces, an interval of the domain of f of length at least $T_{\min}$ over which f has minimum average value can be computed in $O(n)$ time.*

The result on monotone functions immediately implies the desired result on subtrajectory similarity without time shift.

Corollary 1. *Given $T_{\min} > 0$ and two polygonal chains with $O(n)$ vertices each that represent trajectories, the most similar subtrajectories corresponding to the same time interval and of duration at least $T_{\min}$ can be computed in $O(n)$ time.*

4 Approximation algorithms for subtrajectories with time shift

We next direct our attention to the case of similarity with time shifts. In this case an exact solution requires the optimization of functions that have linear description size and whose optimization

cannot be assumed to be possible exactly in reasonable models of computation [15] (see also Section 4.2). Therefore we aim for approximation algorithms.

We consider two versions of the problem: fixed-duration time intervals, and non-fixed duration time intervals where a minimum duration is specified. We obtain different running times in the cases where the observation times (vertices) are regularly-spaced in time and the cases where they are not.

In this section we begin by showing that we can replace the Euclidean distance function by a polygonal convex distance function without much loss in the average distance. Then we discuss the two-dimensional solution space for a fixed-duration time interval. These ideas together lead to polynomial-time $(1+\varepsilon)$-approximation algorithms for the fixed-duration case. Recall that ε is a parameter that must be set beforehand, and determines the quality of the approximation as well as the efficiency of the algorithm.

Then we present a simple technique to reduce the non-fixed duration problem to a number of instances of the fixed-duration problem, and obtain $(1+\varepsilon)$-approximation algorithms for the non-fixed duration case as well.

4.1 Using polygonal convex distance functions

The Euclidean distance between two points p and q can be seen as the scaling factor needed for a unit circle centered at p to have q lie on it. The L_∞-distance between two points is defined similarly, but with an axis-aligned square of side length 2 instead of a unit circle. A well-known idea to obtain $(1+\varepsilon)$-approximation algorithms for geometric problems using Euclidean distance is to replace the distance function d by a polygonal convex distance function $\hat{d}$ defined by a regular $2k$-gon of diameter 2. We will choose $k = \lceil c/\sqrt{\varepsilon} \rceil$ for a suitable constant c. This guarantees that the Euclidean distance between two points is $(1+\varepsilon)$-approximated by the polygonal convex distance function. We immediately observe that our similarity measure, which is an average distance, will also be $(1+\varepsilon)$-approximated if we use $\hat{d}$ instead of d.

Consider two line segments from two trajectories. We can parameterize each by a uniform parametrization of the unit interval. The function $\hat{d}$ that maps two positions, one on each segment, to their distance has as its domain $[0,1] \times [0,1]$.

Lemma 1. *Over its domain, $\hat{d}$ is piecewise linear with $O(1/\sqrt{\varepsilon})$ pieces.*

Lemma 1 follows from Lemma 2, in which the line segments are extended to lines uniformly parameterized over $\mathbb{R}$. For two segments, the domain referred to in Lemma 1 is a parallelogram-shaped part of the domain for lines (which is $\mathbb{R}^2$).

Lemma 2. *Given two lines parametrized over $\mathbb{R}$ with constant speed. Let $\hat{d}$ be a polygonal convex distance using a regular $2k$-gon for these lines defined over the (t, t_{shift})-plane. The function $\hat{d}$ is piece-wise linear. Its domain is subdivided by k lines meeting in a common point. Inside each of the resulting $2k$ cells, $\hat{d}$ is linear.*

Proof. Let τ_1 and τ_2 be parameterizations of the two lines and let p be the intersection point of the lines. Assume that $\tau_1(0) = p = \tau_2(t_0)$ (otherwise re-parametrize). The analytic description of $\hat{d}(t, t_{shift})$ depends on the slope of the line through $\tau_1(t)$ and $\tau_2(t + t_{shift})$. It suffices to show, that for a fixed slope the set of points (t, t_{shift}) that induce this slope or its negative is a line through $(0, t_0)$.

To show this, we formulate τ_1 and τ_2 as

$$\tau_1(t) = p + \lambda_1 t v_1$$
$$\tau_2(t) = p + \lambda_2 (t - t_0) v_2$$

where λ_1, λ_2 are suitable constants and v_1, v_2 suitable unit vectors.

Now assume (t, t_{shift}) and (t', t'_{shift}) induce the same slope. Then by the second intercept theorem

$$\frac{\lambda_1 t}{\lambda_1 t'} = \frac{\lambda_2 (t + t_{shift} - t_0)}{\lambda_2 (t' + t'_{shift} - t_0}$$

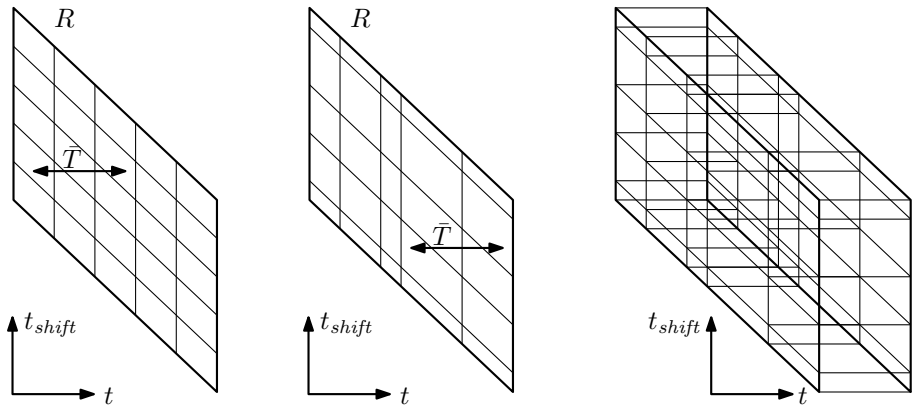


Figure 9: Partition of the domain R of the (t, t_{shift})-plane by regularly and irregularly spaced time observations (left and middle). The placement space, where (t, t_{shift}) represents subtrajectories starting at $\tau_1(t)$ and $\tau_2(t + t_{shift})$ of duration $\bar{T}$ (right).

which implies

$$\frac{t_{shift} - t_0}{t} = \frac{t'_{shift} - t_0}{t'}.$$

This further implies that the line through $(0, t_0)$ and (t, t_{shift}) and the line through $(0, t_0)$ and (t', t'_{shift}) have equal slope and are thus equal (since they also share the point $(0, t_0)$).

The $2k$ vertices of the $2k$-gon result in k lines through $(0, t_0)$ which yield $2k$ cells. Within each cell the distance function $\hat{d}$ is determined by the same side of the $2k$-gon and is therefore linear. □

4.2 Analyzing the placement space

The distance between two points $\tau_1(t)$ and $\tau_2(t+t_{shift})$ depends on the two arguments t and t_{shift}. We therefore consider the (t, t_{shift})-plane. A parallelogram-shaped subregion R of this plane (the domain of d and $\hat{d}$) corresponds to all pairs of locations, one on each of the trajectories. We view d and $\hat{d}$ as bivariate functions over R, determined by the arguments t and t_{shift}. An interval I of duration $\bar{T}$ corresponds to a horizontal line segment of length $\bar{T}$, and if the interval I lies in R, it corresponds to two subtrajectories of duration $\bar{T}$ of the two trajectories τ_1 and τ_2. The image of I under d is a curve in 3-space in a vertical plane, and the average area under this curve (in the vertical plane) is exactly the average distance of the two subtrajectories of duration $\bar{T}$. For the Euclidean distance d, the curve is a piecewise hyperbolic curve, and for the polygonal convex distance function $\hat{d}$, the curve is piecewise linear (polygonal).

Since the trajectories τ_1 and τ_2 consist of line segments, the region R in the (t, t_{shift})-plane is partitioned into a quadratic number of parallelograms, representing when $\tau_1(t)$ and $\tau_2(t + t_{shift})$ lie on the same line segments of τ_1 and τ_2, see Figure 9. The parallelograms are formed by n vertical lines and n lines with slope -1. If the observations are regularly spaced in time, then all parallelograms are of the same size. Let us call two (horizontal) line segments *combinatorially distinct in* R if they intersect a different subset of parallelograms.

Since we are interested in horizontal line segments of length $\bar{T}$, we can use the partitioning of R and the value of $\bar{T}$ to define a *placement space*. Let the left endpoint of a horizontal line segment of length $\bar{T}$ be its reference point. The Minkowski sum of the parallelogram grid and the horizontal segment gives the placement space where each cell corresponds to a combinatorially distinct placement. The Minkowski sum is formed by placing a copy of every vertical line and slope -1 line a distance $\bar{T}$ to the left, and by converting every vertex of the parallelogram grid into a horizontal line segment that extends $\bar{T}$ to the left, see Figure 9 (right). The placement

space is defined by $O(n^2)$ horizontal line segments of length $\bar{T}$ and $O(n)$ vertical lines and lines of slope -1.

Lemma 3. *There are $O(n^2)$ combinatorially distinct placements of a fixed-length horizontal line segment in R if the observations are regularly spaced, and $O(n^3)$ otherwise.*

Proof. Assuming regularly spaced observations, consider the subdivision of R into parallelograms. If we fix the parallelogram containing the left endpoint of the line segment, then there are at most six combinatorially distinct line segments possible. The quadratic bound follows immediately.

To prove the cubic bound, consider the placement space and observe that any of the $O(n^2)$ horizontal segments defining it contains at most $O(n)$ intersections. It follows that the placement space has $O(n^3)$ intersection points. By Euler's formula, its complexity is cubic as well. □

Now, to find the optimal solution, we would need to minimize d for each of the $O(n^2)$ combinatorially distinct placements. However, a description of the function d for one placement does not have constant size in this case. The number of terms of the description is linear in the number of cells crossed, and all terms involve square roots. In the standard models of computation, such functions cannot be minimized. Thus, we cannot compute an exact solution of the minimum distance subtrajectory problem with time shift. Therefore we approximate the function d by $\hat{d}$.

Since we will use the domain R with respect to the polygonal convex distance function $\hat{d}$, each parallelogram is further subdivided by $O(k) = O(1/\sqrt{\varepsilon})$ wedges that we get from the supporting lines of two fixed line segments of the trajectories corresponding to that parallelogram. We call two (horizontal) line segments *apx-combinatorially distinct in R* if they intersect a different subset of cells obtained from the parallelograms subdivided according to the wedges for $\hat{d}$. The subdivision of the parallelograms leads to a more complex placement space.

Lemma 4. *There are $O(n^3/\varepsilon)$ apx-combinatorially distinct placements of a fixed-length horizontal line segment in R if the observations are regularly spaced.*

Proof. We analyze the placement space again and observe that it is an arrangement formed by $O(n^2/\sqrt{\varepsilon})$ horizontal line segments of length $\bar{T}$, $O(n^2/\sqrt{\varepsilon})$ line segments of length at most $\sqrt{5}$ times the observation interval time (the maximum length of any line segment in a parallelogram), and $O(n)$ vertical lines and lines of slope -1.

In any horizontal strip of unit height, there are $O(n/\sqrt{\varepsilon})$ line segments, so the placement space in that strip has complexity $O(n^2/\varepsilon)$. Since R can be covered by $O(n)$ of such strips, the complexity of the placement space is $O(n^3/\varepsilon)$. □

Lemma 5. *There are $O(n^4/\varepsilon)$ apx-combinatorially distinct placements of a fixed-length horizontal line segment in R if the observations are not regularly spaced.*

Proof. The placement space is formed by $O(n^2/\sqrt{\varepsilon})$ line segments as in the proof of Lemma 4, but we do not have the length property nor the strip properties. An arrangement of $O(n^2/\sqrt{\varepsilon})$ line segments has complexity $O(n^4/\varepsilon)$. □

4.3 Algorithms for the fixed-duration case

We can now solve the subtrajectory similarity problem with time shift for fixed-duration time intervals as follows. We first compute the placement space, which is standard by arrangement computation [13]. Then we determine for one cell the function that gives the area below a horizontal interval of length $\bar{T}$, expressed in t and t_{shift}. This function is quadratic, because $\hat{d}$ is a piecewise linear function. Now we can fill in this function for all cells by traversing the placement space and updating the function. Crossing one edge in the placement space leads to an update that can be done in constant time; it can be derived from the edge we cross. In each cell, we minimize the quadratic function and determine the overall minimum. Its coordinates t and t_{shift} define the starting times of the approximately most similar subtrajectories of duration $\bar{T}$. All steps run in time linear in the complexity of the placement space.

Theorem 2. *Let $\varepsilon > 0$ be any fixed constant. The fixed-duration most similar subtrajectory problem with time shift of two trajectories consisting of n line segments can be approximated within a factor $1+\varepsilon$ in $O(n^4/\varepsilon)$ time. If the observation intervals are regularly spaced, the running time is $O(n^3/\varepsilon)$.*

4.4 Reducing non-fixed duration to fixed duration

Suppose that we have an algorithm that solves or approximates the fixed duration problem. We can use it to obtain an approximation for the non-fixed duration version. Recall that there exists an optimal interval of duration less than twice as much as the minimum duration $T_{\min}$ that was specified, because longer intervals can be halved, and one of the halves has an average distance that is no larger.

We will run the fixed-duration algorithm with durations $T_{\min}$, $(1+\frac{\varepsilon}{2})\cdot T_{\min}$, $(1+\varepsilon)\cdot T_{\min}$, $(1+\frac{3\varepsilon}{2})\cdot T_{\min}$, ..., up to a duration that is nearly $2\cdot T_{\min}$ (within $\frac{\varepsilon}{2}\cdot T_{\min}$), and report the solution with the smallest average distance. We claim that if the fixed-duration algorithm is exact, this gives a $(1+\varepsilon)$-approximation of the optimum using $O(1/\varepsilon)$ runs of the fixed-duration algorithm.

Let I^* be the interval with duration T^* that gives the optimum, and let $\tilde{T}$ be the longest duration less or equal to T^* that we used in our algorithm. Then $0 \le T^* - \tilde{T} \le \frac{\varepsilon}{2}T_{\min}$. The area $\tilde{A}$ below the horizontal line segment of length $\tilde{T}$ is at most the area A^* below the line segment corresponding to I^*. So for the average distance of the approximate solution assuming $\varepsilon \le 1$ we have:

$$\frac{\tilde{A}}{\tilde{T}} \le \frac{A^*}{T^* - \frac{\varepsilon}{2}T_{\min}} \le \frac{A^*}{T^* - \frac{\varepsilon}{2}T^*} \le (1+\varepsilon)\frac{A^*}{T^*} .$$

Now we combine the approximation of the fixed-duration algorithm and the reduction of the non-fixed duration problem to the fixed duration problem. Let a fixed $0 < \varepsilon \le 1$ be given. We use $\varepsilon/3$ in both approximations and obtain an overall approximation with a factor $\left(1+\frac{\varepsilon}{3}\right)\left(1+\frac{\varepsilon}{3}\right) \le 1 + \frac{2\varepsilon}{3} + \frac{\varepsilon^2}{9} \le 1+\varepsilon$.

Theorem 3. *Let $\varepsilon > 0$ be any fixed constant. The most similar subtrajectory problem with time shift and duration at least $T_{\min}$ of two trajectories consisting of n line segments can be approximated within a factor $1+\varepsilon$ in $O(n^4/\varepsilon^2)$ time. If the observation intervals are regularly spaced, the running time is $O(n^3/\varepsilon^2)$.*

5 Conclusion

In this paper, we considered finding similar subtrajectories using a distance measure that is defined as the average Euclidean distance at corresponding times. For equal starting times, we gave a linear-time algorithm, and for not necessarily equal starting times, we gave $(1+\varepsilon)$-approximation algorithms. The latter are not very efficient, and it is an open problem whether more efficient solutions exist. This may be possible in general, or under realistic input assumptions. It may also be that the algorithms are more efficient in practice than the worst-case running time suggests. One can expect this to be the case when the subtrajectories to be computed are short with respect to the whole trajectory, as this reduces the complexity of the placement space. Another way to cope with the relative inefficiency of some of our algorithms is by trajectory simplification [1, 6, 11] as a preprocessing step, to reduce the trajectories to no more than a few thousand vertices each. It would be interesting, to experimentally test the efficiency of our algorithms.

Our algorithms can of course be used for multiple trajectories by testing two trajectories at a time. For large sets of trajectories with many vertices, this may become too inefficient in practice. It is of significant interest to develop algorithms and data structures that avoid testing all pairs of trajectories, but can still find two subtrajectories of two trajectories that are most similar.

Another direction for future research is to determine for various types of trajectory data whether our similarity measure is appropriate for clustering purposes. It may well be that the geometric and time-dependent similarity, as studied in this paper, must be combined with more

factors to obtain good clustering results for specific applications. Hence, it is of significant interest to develop distance measures that take further attributes of the trajectories into account, where the selection of these attributes depend on the application.

Acknowledgement

We thank Günter Rote for pointing us to the connection between the maximum density segment problem and convex hulls.

References

[1] G. Barequet, D. Z. Chen, O. Daescu, M. T. Goodrich, and J. Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. *Algorithmica*, 33:150–167, 2002.

[2] T. Bernholt, F. Eisenbrand, and T. Hofmeister. A geometric framework for solving subsequence problems in computational biology efficiently. In *Proc. 23rd ACM Symp. on Comput. Geom. (SOCG'07)*, 310–318, 2007.

[3] K. Buchin, M. Buchin, and J. Gudmundsson. Constrained free space diagrams: a tool for trajectory analysis. *International Journal of Geographical Information Science*, 24:1101–1125, 2010.

[4] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting Commuting Patterns by Clustering Subtrajectories. To appear in *Int. J. Comput. Geom. and Appl., Special Issue: Selected Papers from the 19th Int. Symp. on Algorithms and Computation (ISAAC'08)*.

[5] K. Buchin, M. Buchin, and Y. Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*, 645–654, 2009.

[6] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal*, 15:211–228, 2006.

[7] K.-M. Chung and H.-I. Lu. An optimal algorithm for the maximum-density segment problem. *SIAM J. Comput.*, 34:373–387, 2005.

[8] L. Chen, M. Tamer Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proc. 2005 ACM SIGMOD Int. Conf. on Management of data (SIGMOD'05)*, 491–502, 2005.

[9] E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-based Most Similar Trajectory Search. In *Proc. IEEE 23rd Int. Conf. on Data Engineering*, 816–825, 2007.

[10] M. H. Goldwasser, M.-Y. Kao, and H.-I. Lu. Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. *J. Comput. Syst. Sci.*, 70:128–144, 2005.

[11] J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle. Compressing spatio-temporal trajectories. In *Proc. Int. Symp. on Algorithms and Computation (ISAAC'07)*, 763–775, 2007.

[12] J. Gudmundsson, P. Laube, and T. Wolle. Movement Patterns in Spatio-Temporal Data. In: S. Shekhar and H. Xiong, editors, *Encyclopedia of GIS*, Springer, Heidelberg, 726-732, 2008.

[13] D. Halperin. Arrangements. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, 389–412, CRC Press, 2nd edition, 2004.

[14] E. J. Keogh and M. J. Pazzani. Scaling up Dynamic Time Warping for Datamining Applications. In *Proc. 4th Asia-Pacific Conf. on Knowledge Discovery and Data Mining (PADKK'00)*, 285–289, 2000.

[15] M. van Kreveld and J. Luo. The definition and computation of trajectory and subtrajectory similarity. In *Proc. 15th ACM Int. Symp. on Geographic Information Systems (ACM-GIS'07)*, 44–47, 2007.

[16] J. Lee, J. Han, X. Li, and H. Gonzalez. *TraClass*: Trajectory classification using hierarchical region-based and trajecory-based clustering. *Proceedings of the VLDB Endowment* 1:1081–1094, 2008.

[17] J. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. 2007 ACM SIGMOD International Conference on Management of data (SIGMOD'07).*, 593–604, 2007.

[18] Y.-L. Lin, T. Jiang, and K.-M. Chao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *J. Comput. Syst. Sci.*, 65:570–586, 2002.

[19] M. Nanni and D. Pedreschi. Time-focused clustering of trajectories of moving objects. *J. Intell. Inf. Syst.*, 27:285–289, 2006.

[20] G. Sinha and D.M. Mark. Measuring similarity between geospatial lifelines in studies of environment health. *J. Geographical Systems*, 7:115–136, 2005.

[21] G. Trajcevski, H. Ding, P. Scheuermann, R. Tamassia, and D. Vaccaro. Dynamics-aware similarity of moving objects trajectories. In *Proc. 15th ACM Int. Symp. on Geographic Information Systems (ACM-GIS'07)*, 1–8, 2007.

[22] M. Vlachos, G. Kollios and D. Gunopulous. Discovering similar multidimensional trajectories. *J. Intelligent Information Systems*, 27:267–289, 2006.

[23] M. Vlachos, D. Gunopulos, and G. Das. Rotation invariant distance measures for trajectories In *Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'04)*, 707–712, 2004.

[24] Y. Yanagisawa, J. Akahani and T. Satoh. Shape-based similarity query for trajectory of mobile objects. In *Proc. 4th Int. Conf. on Mobile Data Management (MDM'03)*, 63–77, 2003.