Finding a minimum stretch of a function

Finding a Minimum Stretch of a Function[*]

Kevin Buchin[†] Maike Buchin[*] Marc van Kreveld[*] Jun Luo[‡]

Abstract

Given a piecewise monotone function $f\colon \mathbb{R} \to \mathbb{R}$ and a real value $T_{\min}$, we develop an algorithm that finds an interval of length at least $T_{\min}$ for which the average value of f is minimized. The run-time of the algorithm is linear in the number of monotone pieces of f if certain operations are available in constant time for f. We use this algorithm to solve a basic problem arising in the analysis of trajectories: Finding the most similar subtrajectories of two given trajectories, provided that the duration is at least $T_{\min}$. Since the precise solution requires complex operations, we also give a simple $(1+\varepsilon)$-approximation algorithm in which these operations are not needed.

1 Introduction

Where does a function f have its extremes? If we look at f at a larger scale, a more useful answer to this question than the singular extrema of f may be high and low "plateaus" of f. Therefore, we consider the problem of finding an interval of the domain of f for which the average of f is minimum. The interval should have at least a given length, otherwise its length would always be zero. We develop an algorithm for this problem for functions which are piecewise monotone. The run-time of the algorithm is linear in the number of monotone pieces of f. The straightforward algorithm of iterating over all possible start and end pieces, using some precomputed values, and optimizing, would have quadratic run-time.

Our study of this problem is motivated by geometric problems occurring in geographic data analysis, in particular, the problem of finding similar subtrajectories of moving objects [6]. Given two trajectories, we wish to determine a time interval of at least a certain length such that the trajectories are close during that time interval. By "close" we mean that the average distance during the time interval is as small as possible. This application, however, is an instance of the more general problem we are solving in this paper. Another geographic application we are interested in is the following. Assume a moving object measuring some quantity while it moves, for instance, the height. We want to find high (or low) plateaus of this quantity. There are more ways to measure the similarity of trajectories, see for example the references given in [6].

The discrete version of the minimum stretch problem occurs in biological sequences alignment and has been studied there. Several similar linear-time algorithms have been given [2, 4, 5], which provide the basis of the ideas used in our algorithm. Our algorithm is considerably more complex, however, due to allowing any start and end point of the interval, and allowing any type of piecewise monotone function. For the discrete sequence version there is also a geometric algorithm using very different ideas [1].

2 Algorithm

In this section, we develop an algorithm that minimizes the average height over intervals of a piecewise monotone function f, where the interval has a non-fixed duration $T \geq T_{\min}$. The run-time of the algorithm is linear in the number of pieces of f.

Fixed length. Before we give the algorithm, we consider a simple version of the problem where the length of the interval is fixed to be $\hat{T}$. For this problem, a trivial linear-time algorithm exists by scanning the function and maintaining its average. The solution with $\hat{T} = T_{\min}$ is a 2-approximation for the problem with non-fixed length, assuming f is nonnegative. To see this, note that always an optimal length T with $T_{\min} \leq T < 2T_{\min}$ exists (for larger T, split in the middle and choose a half with smaller or equal average). If the interval I_{opt}, with smallest average, has a duration T' with $T_{\min} \leq T' \leq 2T_{\min}$, then the dissimilarity for any subinterval of length $T_{\min}$ is larger by at most a factor $T'/T_{\min} \leq 2$.

Furthermore, the factor two can be obtained in the limit, as demonstrated in Figure 1: the total duration is $2T_{\min} - \varepsilon$ and the distance between the trajectories is 0 except for a duration of ε in the middle (for illustration purposes, they are shown with a small vertical offset). In this example, fixed and non-fixed duration differ by a factor of $(2T_{\min} - \varepsilon)/T_{\min}$ which is $2 - \varepsilon$ for $T_{\min} = 1$.

[*]This research has been funded by the Netherlands' Organisation for Scientific Research (NWO) under BRICKS/FOCUS project no. 642.065.503 and by the German Research Foundation (DFG) under grant BU 2419/1-1.

[†]Department of Information and Computing Sciences, Utrecht University, 3584CH Utrecht, The Netherlands, {buchin, maike, marc}@cs.uu.nl

[‡]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, China, jun.luo@sub.siat.ac.cn

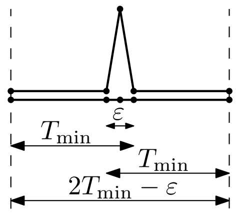


Figure 1: Worst-case ratio example.

If we run the fixed duration algorithm with durations $T_{\min}$, $(1+\varepsilon)\cdot T_{\min}$, $(1+2\varepsilon)\cdot T_{\min}$, $\ldots$, $2\cdot T_{\min}$, and take the overall optimum, then we have a $(1+\varepsilon)$-approximation algorithm for the general problem (assuming f is nonnegative) that runs in $O(n/\varepsilon)$ time.

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, and 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. We may want to start the interval earlier, to lower the average value of f over the chosen interval (see Figure 2). 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

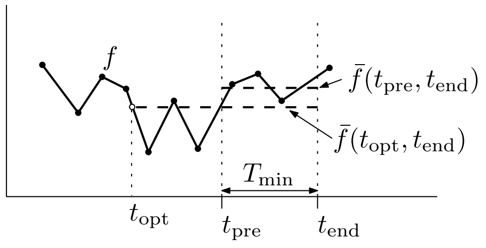
$$\bar{f}(I) := \bar{f}(t', t'') = \frac{\int_{t'}^{t''} f(t)dt}{t'' - t'} .$$

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 valid if t_{end} is fixed.

For description purposes, we fix t_{end} and therefore t_{pre} for the moment. Then interval $[t_{\text{pre}}, t_{\text{end}}]$ 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 $\bar{f}(.)$. Even if the function value of f just before t_{pre} is greater than $\bar{f}(t_{\text{pre}})$, then extending the interval to



f
$\bar{f}(t_{\text{pre}}, t_{\text{end}})$
$\bar{f}(t_{\text{opt}}, t_{\text{end}})$
$T_{\min}$
t_{opt}
t_{pre}
t_{end}

Figure 2: Averages of f over $[t_{\text{pre}}, t_{\text{end}}]$ and $[t_{\text{opt}}, t_{\text{end}}]$.

a starting time (sufficiently far) before t_{pre} may still give a lower average. In Figure 2 we observe that the optimal starting time $t_{\text{opt}} \le t_{\text{pre}}$, given t_{end} as the end of the interval, 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 the value of $\bar{f}(t_{\text{pre}})$ only 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 have to go further back with t_{opt}, but never forward).

Assumptions. To compute the minimum stretch in linear time we need to 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 $F(a, s) = as + b$, where $F(a, s) = \int_a^s f(t)dt$.
3. Find a stretch of minimum average value, if the monotone pieces for the left and the right endpoint of the stretch are given and the integral of f for the intervals in between has been evaluated.

For simplicity, we will assume that f is continuous. We can extend the ideas to handle non-continuous functions, but the definitions and description of the method become more tedious.

Data structure. Let f be a piecewise monotone function with break points $t_1, \ldots, t_n$, that is, f is monotone in between each pair t_i and t_{i+1} for $1 \le i < n$. At all times our data structure consists of the interval $I_0 = [t_{\text{pre}}, t_{\text{end}}]$ and a set of intervals $I_1, \ldots, I_m$, where $I_i = [s_i, s_{i-1}]$, for $i = 1, \ldots, m$, $m \ge 0$, and $s_m < s_{m-1} < \ldots < s_1 < s_0 = t_{\text{pre}}$. To define $s_1, \ldots, s_m$ and m, we first define a function $l(s)$ which, intuitively, tells how far to the left we can always extend an interval if we extend 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

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

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 (to its left).

We can now define the s_i, $1 \le i \le m$, by

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

Thus, if $l(s_{i-1}) = s_{i-1}$, then we set s_i either to the next breakpoint 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 $I_1, \dots, I_m$: 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 for all $s' \in [s_i, s_{i-1}]$ we have $f(s_{i-1}) = f(s_i) = \bar{f}(I_i)$.
2. If I_i is complete, then for all $s' \in (s_i, s_{i-1})$ we have $\bar{f}(s_i) < \bar{f}(s')$, and also I_{i+1} is decreasing if $i \ge 1$.
3. $\bar{f}(I_i) < \bar{f}(I_j)$ 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 second 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 height of f over intervals in the data structure. We will not need to consider intervals at the left end of our data structure if their (partial) inclusion would increase the average height. It follows that the last interval I_m that we need is a decreasing interval. Also, we do not need intervals further to the left if their inclusion would result in an average height which is larger than a previously found average height. Hence, we will have:

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

Our data structure maintains the sequence of break points $t_{\text{end}}, s_0, s_1, \dots, s_m$, the pieces of f that contains each, and the sequence $F(I_0), \dots, F(I_m)$, 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 $F(I_{m-1} \cup \dots \cup I_1)$.

Algorithm. We can find the interval with minimum average height as follows. We scan with the interval $[t_{\text{pre}}, t_{\text{end}}]$ from start to end along the domain of the function f, and maintain the information we just described. Most of this information can only change 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 $\tilde{t}_{\text{end}}$, $\tilde{t}_{\text{pre}} = \tilde{s}_0$, $\tilde{s}_1$, $\tilde{I}_0$, etc., to denote values in between events t_{end} and t'_{end}.

In between two consecutive event points $t_{\text{end}} \le \tilde{t} \le t'_{\text{end}}$, we need to minimize $\bar{f}(t, \tilde{t})$ over the choices of t and $\tilde{t}$ with $t \le \tilde{t} - T_{\min}$, where we know 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 $F(\tilde{I}_{m-1} \cup \dots \cup \tilde{I}_0) = F(I_{m-1} \cup \dots \cup I_3 \cup \tilde{I}_2 \cup \tilde{I}_1 \cup \tilde{I}_0)$ 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. Furthermore, we maintained $F(I_{m-1} \cup \dots \cup I_1)$ and the $F(I_i)$ at the previous event point t_{end}, and $\tilde{t}$ does not pass any vertex of f before the next event, so we can derive the expression $F(\tilde{I}_{m-1} \cup \dots \cup \tilde{I}_0)$ 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.

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

2. 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 immediately).

3. 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.

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

Note that instead of stopping at events of type 4, we can also check if they happened at the next event of type 1, 2, or 3.

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. The event points of types 2, 3, and 4 are detected 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 t of type 2 occurs for $t_{\text{end}} < t < t'_{\text{end}}$ if $f(s_2) = \bar{f}(s_2, t - T_{\min})$. To detect this, we observe

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

and make an expression in t. This takes constant time using the values $F(I_2)$, $F(I_1)$, and $F(I_0)$. Then we find t by setting it equal to $f(s_2)$ (using the second assumption). Events of type 3 are detected in a similar manner.

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

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

which we can compute in constant time as before.

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 can be done in constant time.

Correctness and run-time. The optimal solution is the minimal value $\bar{f}(t, \tilde{t})$ for $t \leq \tilde{t} - T_{\min}$. Assume $(t, \tilde{t})$ is such an optimal pair where $\tilde{t}$ is minimal such that it is the second part of an optimal pair and t is maximal such that it is the first part of an optimal pair with $\tilde{t}$. Let $t_{\text{end}} < \tilde{t} < t'_{\text{end}}$ be the event points left and right of $\tilde{t}$. We need to prove that $t \in I_m$.

Suppose $t < s_m$. Then t lies in an interval previously discarded. Let t'' be the right endpoint of this interval before it was discarded. Since the interval was discarded there are $\hat{t}', \hat{t}$ with $\hat{t}' \leq \hat{t} - T_{\min}$ and $\hat{t} \leq \tilde{t}$ such that $\bar{f}(t, t'') \geq \bar{f}(\hat{t}', \hat{t})$. 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 $s_{i-1} \geq t > s_i$ for some $i \leq m-1$. But then including the interval from s_i to t to $(t, \tilde{t})$ would decrease the average because I_m was not discarded, contradicting the optimality of $(t, \tilde{t})$.

It is not hard to see that the number of events is linear, and the running time is $O(n)$.

3 On trajectory similarity

A (time-dependent) trajectory is a continuous function from $[0, 1]$ to the plane. We use the algorithm above to solve: Given two piecewise linear trajectories τ_1, τ_2 and $T_{\min} > 0$, find a time interval $[t_1, t_2]$ of length $\geq T_{\min}$ that minimizes the average distance $\int_{t_1}^{t_2} d(\tau_1(t), \tau_2(t))dt$, where d is the Euclidean distance.

On an interval on which both τ_1 and τ_2 are linear, $d(\tau_1(t), \tau_2(t)) = \sqrt{At^2 + Bt + c}$, which corresponds to a hyperbolic arc. It has no local maxima and possibly one local minimum interior to the interval. We split the intervals at such minima, so the distance between the trajectories is a piecewise monotone function and we can apply the algorithm above. It remains to see whether the operations needed for the algorithm above are available for $d(\tau_1(t), \tau_2(t))$. For a continuous function $f(t)$ (such as $d(\tau_1(t), \tau_2(t))$), a minimizing stretch $[t_1, t_2]$ with $t_2 - t_1 \geq T_{\min}$ falls in one of the following cases: $t_2 - t_1 = T_{\min}$ or $t_1 = 0$ or $t_2 = 1$ or $f(t_1) = f(t_2) = \bar{f}(t_1, t_2)$. This gives an equation in, say, t_1 such that any left endpoint of a minimizing interval is a solution to the equation. This improves a quadratic time solution in [6].

The precise solution for $d(\tau_1(t), \tau_2(t))$ requires fairly complicated operations. A $(1 + \varepsilon)$-approximation can be obtained by replacing the Euclidean distance by a polyhedral distance function. We use a regular k-gon with $k = O(\sqrt{1/\varepsilon})$ vertices [3] to define it. In an interval in which both $\tau_1(t)$ and $\tau_2(t)$ are linear, this results in a piecewise linear distance function with $O(\sqrt{1/\varepsilon})$ pieces. The total run-time is then $O(n/\sqrt{\varepsilon})$.

We note that with polyhedral distance functions we can find approximate solutions for trajectory similarity with time shifts [6].

References

[1] 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.*, pages 310–318, 2007.

[2] K.-M. Chung and H.-I. Lu. An optimal algorithm for the maximum-density segment problem. *SIAM J. Comput.*, 34(2):373–387, 2005.

[3] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approximation Theory*, 10:227–236, 1974.

[4] 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(2):128–144, 2005.

[5] 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(3):570–586, 2002.

[6] M. van Kreveld and J. Luo. The definition and computation of trajectory and subtrajectory similarity. In *Proc. 15th ACM Symp. on Advances in GIS*, pages 44–47. ACM, 2007.