Linear Time Algorithm for Optimal Feed-link Placement

Marko Savić[*] Miloš Stojaković[*†]

Abstract

Given a polygon representing a transportation network together with a point p in its interior, we aim to extend the network by inserting a line segment, called feed-link, which connects p to the boundary of the polygon. Once a feed link is fixed, the geometric dilation of some point q on the boundary is the ratio between the length of the shortest path from p to q through the extended network, and their Euclidean distance. The utility of a feed-link is inversely proportional to the maximal dilation over all boundary points.

We give a linear time algorithm for computing the feed-link with the minimum overall dilation, thus improving upon the previously known algorithm of complexity that is roughly $O(n \log n)$.

1 Introduction

For our purposes, a network P is an embedding of a connected graph into two-dimensional Euclidean space. Given two points p and q (on edges or vertices) of P, their *network distance* is defined as the length of the shortest curve contained in P connecting p and q. We define the *dilation* (sometimes also called the *detour*, or slightly less formally the *crow flight conversion coefficient*) as the ratio of the network distance between points p and q, and their Euclidean (crow flight) distance. The *geometric dilation* of the network P is the maximum detour taken over all pairs of points on the network.

Given a network P and a point p not on P, we want to extend the network by adding a single line segment, called *feed-link*, connecting p with a point on P. Note that a feed-link may have more than one point in the intersection with P, but we do not regard these points as connection points. An optimal feed-link is the one that minimizes the maximum dilation from the point p to a point on P.

We solve the problem of finding an optimal feed-link in polygonal networks by constructing an algorithm which runs in linear time in the size of the polygon.

Background and related work. In applications, we often encounter networks as models to real world structures, such as road or subway networks. It happens in geographical information systems that the locations of settlements are provided, but the data describing

[*]Department of Mathematics and Informatics, University of Novi Sad, Serbia. Partly supported by Ministry of Education and Science, Republic of Serbia. E-mail: {*marko.savic, milos.stojakovic*}*@dmi.uns.ac.rs*

[†]Partly supported by Provincial Secretariat for Science, Province of Vojvodina.

roads is only partial (e.g., only the location of larger roads are known, while smaller roads are missing from the database). However, in order to perform various network analysis a network needs to be connected, i.e. every settlement needs to lie on a road, which motivates us to find a way to extend the network so that the disconnected nodes become attached.

Depending on the requirements, there are many ways to connect a new node to an existing network. Probably the most straightforward one is to simply snap the location to the closest point on the network. This may be unsuitable as the node location is modified. Also, it can happen that two points geometrically close to each other are snapped to parts of network that are far away, which may be undesirable. Another approach is to link all new nodes inside a network face to the *feed-node* which is then connected to the network. Alternatively, each new node can be individually attached to the network using a *feed-link*. This approach was taken, e.g., in [1, 2], where the new location is simply connected to the nearest existing location by a feed-link.

In an attempt to reduce unnecessary detours, Aronov et al. in [3] introduced a more sophisticated way of choosing where on the existing network to attach the new feed-link, using the dilation as the measure of feed-link quality. They presented an algorithm to compute the feed-link achieving minimal dilation with a running time of $O(\lambda_7(n)\log n)$, where n is the number of vertices on the boundary of the face, and $\lambda_7(n)$ is the maximum length of a Davenport-Schinzel sequence of order 7 on n symbols, a slightly superlinear function. We consider this same setting and make an improvement by presenting an $O(n)$ time algorithm.

Similar criterion for choosing the feed-link is recently considered by Bose at al. in [4]. They solve the problem of finding the *minimum eccentricity* feed-link, which minimizes the largest network distance from the new point to any point on the network.

Several results can be found in the literature analyzing the dilation and the stretch factor (the largest dilation between a pair of network nodes) of a given network. First result on that topic is given by Narasimhan and Smid in [5], where an approximate algorithm for computing the stretch factor is given. Ebbers-Baumann at al. in [6] give an approximate algorithm for computing the dilation of planar polygonal chains, and Agarwal at al. in [7] give exact algorithms for computing both the dilation and the stretch factor of planar polygonal paths, trees and cycles. For a result on how to construct a network with a small dilation containing given points, we refer the reader to [8].

Structure and results. In the present paper, we give a linear time algorithm which finds the optimal feed-link – the one that minimizes the maximal dilation to the points on the boundary of the polygon. Although the initial problem statement given in [3] assumes that p lies inside the polygon and the polygon is simple, all of our calculations work out exactly the same for an arbitrary point p in the plane and arbitrary polygons, possibly self-intersecting, and hence, our results hold also in that more general setting.

Our algorithm is based on the idea that finding the maximum dilation can be reduced to finding the minimum slope of a certain line defined on the plot of the distance function from p to the boundary of P. This idea is then employed to construct a sweep algorithm over the points on the boundary.

The rest of the paper is organized as follows. In Section 2 we introduce the notation and give a formal definition of the problem. Then we split the problem into two symmetrical components, the left and the right dilation, so that we could perform our further analysis only on one of them. An alternative view of the problem is given in Section 3 by plotting the distance function from p to points on the boundary of P, and mapping feed-links to levers – line segments defined on the plot, with slope inversely proportional to the left dilation of a feed-link. In Section 4 we design a sweep algorithm which simulates moving of the lever, and outputs a sequence describing different states the lever passes through while moving. Finally, the sweep algorithm is run once for the left and once for the right dilation, and the two produced outputs are then combined to obtain the solution for the original problem. This merging process is described in Section 5.

2 Notation and problem statement

A polygon, which is not necessarily simple, is given as a list of its vertices $v_0, v_1, \ldots, v_{n-1}$. We denote the boundary of that polygon by P. We are also given a point p, not necessarily inside the polygon. A *feed-link* is a line segment pq, connecting p with some point $q \in P$.

For any two points $q, r \in P$ *dilation of r via q* is defined as

$$\delta_q(r) = \frac{|pq| + \operatorname{dist}(q, r)}{|pr|},$$

where $\operatorname{dist}(q, r)$ is the length of the shortest route between q and r over the polygon's boundary, and $|ab|$ is the Euclidean distance between points a and b, see Figure 1.

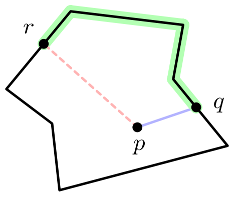


Figure 1: The concept of dilation.

For a point $q \in P$, *dilation via q* is defined as

$$\tilde{\delta}_q = \max_{r \in P} \delta_q(r).$$

The problem of finding the optimal feed-link is to find q such that $\tilde{\delta}_q$ is minimized.

2.1 Left and right dilation

Given two points $q, r \in P$, $P[q, r]$ is the portion of P obtained by going from q to r around the polygon in the positive (counterclockwise) direction, including the points q and r. Let $\mu(q, r)$ be the length of $P[q, r]$, and $\mu(P)$ the perimeter of P.

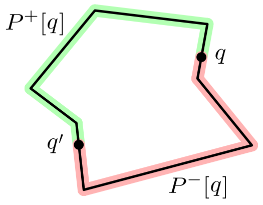


Figure 2: The left and the right portion of P observed from the point q.

For the given $q \in P$, let q' be the point on P, for which $\mu(q, q') = \mu(q', q) = \mu(P)/2$, see Figure 2. By $P^+[q]$ we denote $P[q, q']$, and by $P^-[q]$ we denote $P[q', q]$. Obviously, $P^+[q] \cup P^-[q] = P$ and $P^+[q] \cap P^-[q] = \{q, q'\}$.

For given points $q \in P$ and $r \in P^+[q]$, the *left dilation of* r *via* q is defined as

$$\delta_q^+(r) = \frac{|pq| + \mu(q, r)}{|pr|}.$$

On the other hand, for $r \in P^-[q]$, the *right dilation of* r *via* q is defined as

$$\delta_q^-(r) = \frac{|pq| + \mu(r, q)}{|pr|}.$$

When measuring $\text{dist}(q, r)$, the shortest path from q to r over P must lie entirely either in $P^+[q]$ or $P^-[q]$. This allows us to express the dilation of r via q using the left and the right dilation of r via q

$$\delta_q(r) = \begin{cases} \delta_q^+(r), & \text{if } r \in P^+[q] \\ \delta_q^-(r), & \text{if } r \in P^-[q] \end{cases}.$$

Given point $q \in P$, the *left dilation via* q is defined as $\tilde{\delta}_q^+ = \max_{r \in P^+[q]} \delta_q^+(r)$, and the *right dilation via* q as $\tilde{\delta}_q^- = \max_{r \in P^-[q]} \delta_q^-(r)$. Finally, the dilation via q can be expressed as

$$\tilde{\delta}_q = \max(\tilde{\delta}_q^+, \tilde{\delta}_q^-) = \max_{r \in P} \delta_q(r). \tag{1}$$

In the following two sections we will be concerned only with the left dilation, as the problem of finding the right dilation is, by definition, analogous. In Section 5 we will show how to combine our findings about the left and the right dilation to provide the answer to the original question. To simplify the notation, we omit the superscript + in Section 3 and Section 4, assuming that we deal with the left dilation.

3 Another view of the problem

Let us once again state the setting of our problem. Given a polygon boundary P and a point p, we want to compute the minimum dilation. In the previous section we saw how the dilation can be expressed in terms of the left and the right dilation. In this section, we are going to look at the distance function from p to the points on P, and observe how this function relates to the left dilation. These observations will enable us to work only with the distance function, instead of working with the polygon itself.

First, we parametrize points on P by defining $P(t)$, $t \in \mathbb{R}$, to be the point on P for which $\mu(v_0, P(t)) \equiv t \pmod{\mu(P)}$, see Figure 3.

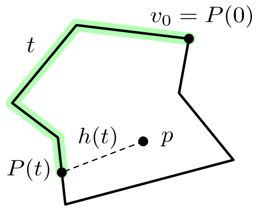


Figure 3: Parametrization of P.

The distance of a point to the points on a straight line is known to be a hyperbolic function. The plot of the distance function $h(t) := |pP(t)|$ is an infinite sequence of hyperbola segments joined at their endpoints, where $(kn + r)$-th hyperbola segment corresponds to the r-th side of P, for $r \in \{0, 1, \ldots, n-1\}$ and $k \in \mathbb{Z}$, see Figure 4. For each $i = kn + r$, the hyperbola h_i is of the form $h_i(t) = \sqrt{(t - m_i)^2 + d_i^2}$, for some values m_i and d_i, so that $m_{kn+r} = m_r + k\mu(P)$, $d_{kn+r} = d_r$, and d_r is the distance between p and the line containing the r-th side of P. The left endpoint of the i-th hyperbola segment is $E_i := (e_i, h(e_i))$, where $e_i = k\mu(P) + \mu(v_r)$, and the right endpoint is at $(e_{i+1}, h(e_{i+1}))$. We will consider that each hyperbola segment contains its left endpoint, but not the right endpoint. By $H(t)$ we denote the point on the plot of h corresponding to the parameter t, so $H(t) := (t, h(t))$. The plot is, obviously, periodic, with the period of $\mu(P)$, that is, $H(t) = H(t + k\mu(P))$. We denote the i-th hyperbola segment with $\mathcal{H}_i$.

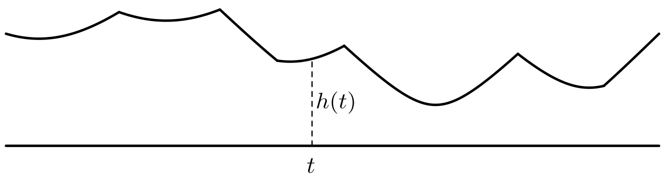


Figure 4: Plot of $h(t)$.

Let $o(t) = t - h(t)$, and $O(t) = (o(t), 0)$, see Figure 5. We also define $o_i(t) = t - h_i(t)$, and $O_i(t) = (o_i(t), 0)$. Given points $q \in P$ and $r \in P^+[q]$, we have their corresponding parameters t_q and t_r, such that $t_q \le t_r \le t_q + \mu(P)/2$. The slope of the line passing through the points $O(t_q)$ and $H(t_r)$ is

$$\begin{aligned} s(t_q, t_r) &:= \operatorname{slope}\left(\ell(O(t_q), H(t_r))\right) \\ &= \frac{h(t_r)}{t_r - t_q + h(t_q)} \\ &= \frac{|pP(t_r)|}{\mu(P(t_q), P(t_r)) + |pP(t_q)|} \\ &= \frac{1}{\delta^+_{P(t_q)}(P(t_r))}, \end{aligned} \tag{2}$$

hence the slope between $O(t_q)$ and $H(t_r)$ is equal to the inverse of the left dilation of r via q.

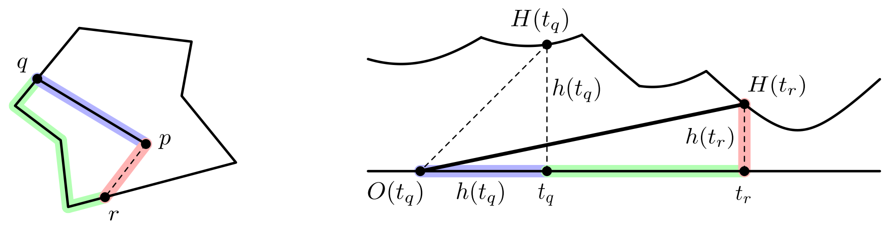


Figure 5: Dilation and slope relation.

We define $\tilde{s}(t_q)$ to be the lowest slope from $O(t_q)$ to $H(t_r)$ among all $t_r \in [t_q, t_q + \mu(P)/2]$. From the previous observation it follows that this slope equals the inverse of the left dilation via q,

$$\begin{aligned} \tilde{s}(t_q) &:= \min_{t_r \in [t_q, t_q+\mu(P)/2]} s(t_q, t_r) \\ &= \min_{t_r \in [t_q, t_q+\mu(P)/2]} \frac{1}{\delta^+_{P(t_q)}(P(t_r))} \\ &= \frac{1}{\max_{r \in P^+[q]} \delta^+_q(r)} \\ &= \frac{1}{\tilde{\delta}^+_q}. \end{aligned} \tag{3}$$

Obviously, $\tilde{s}(t) \in (0, 1]$ because it is strictly positive and $\tilde{s}(t) \leq s(t, t) = 1$. This enables us to estimate the dilation by looking at the slope of the line we just defined.

Lemma 1. *For any two distinct values of t_1 and t_2,*

$$|h(t_2) - h(t_1)| \leq |t_2 - t_1|.$$

Proof. From the triangle inequality we have $|h(t_2) - h(t_1)| \leq |P(t_1)P(t_2)|$, and from the parametrization by distance along P it follows that $|P(t_1)P(t_2)| \leq |t_2 - t_1|$. These inequalities readily imply the statement of the lemma. $\square$

So far, $\tilde{s}(t_q)$ was defined as the minimum only among the slopes $s(t_q, t_r)$ where t_r belongs to the interval $[t_q, t_q + \mu(P)/2]$. However, from Lemma 1 follows that $s(t_q, t_r)$ cannot be less than 1 when $t_r \in [o(t_q), t_q]$, and $\tilde{s}(t_q)$ is at most 1, so this interval can be extended, and we have

$$\tilde{s}(t_q) = \min_{t_r \in [o(t_q), t_q+\mu(P)/2]} s(t_q, t_r).$$

This extension of the interval enables us to define the *lever* and to construct the following *sliding lever algorithm* without having to worry about certain unwanted cases.

4 Sliding lever algorithm

In this section we define the *lever*, the central object of our analysis, as well as the possible states that the lever can have. We want to simulate the movement of the lever, so we analyze all the events that lead to state changes. That allows us to construct a sweep algorithm; however, the most straightforward algorithm will not run in linear time. To fix that we design an additional algorithm to precompute the set of the so-called *retargeting positions*, which we use for certain helper events (*jump destination change events*).

4.1 Lever

For a fixed t, consider the line segment having slope $\tilde{s}(t)$, with one endpoint at $O(t)$ and the other at $(t + \mu(P)/2, \tilde{s}(t)(\mu(P)/2 + h(t)))$, see Figure 6. Let us call that line segment the *lever for* t. Note that the lever only touches the plot, never intersecting it properly.

Let $C(t)$ be the leftmost point in which the lever for t touches the plot, and let $c(t)$ be such that $H(c(t)) = C(t)$. Then $c(t) \in [o(t), t + \mu(P)/2]$ and it is the lowest value in this interval for which $\tilde{s}(t) = s(t, c(t))$. Coming back to the original setup, this means that the left dilation via $P(t)$ reaches its maximum for $P(c(t))$.

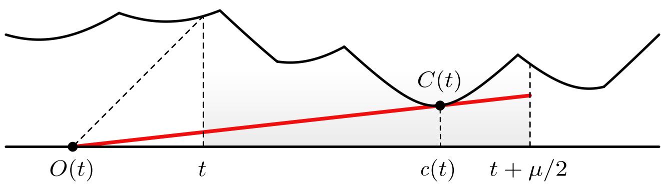


Figure 6: Lever.

We now continuously decrease parameter t and observe what is happening with the updated lever. The following monotonicity lemma states that when t is decreasing $o(t)$ and $c(t)$ are decreasing as well, which means that decreasing t corresponds to "dragging" the lever in the leftward direction.

Lemma 2. *For $t_1 < t_2$ we have $o(t_1) \leq o(t_2)$ and $c(t_1) \leq c(t_2)$.*

Proof. Suppose $t_1 < t_2$. Using Lemma 1 we get

$$\begin{aligned} h(t_2) - h(t_1) &\leq t_2 - t_1, \\ t_1 - h(t_1) &\leq t_2 - h(t_2), \\ o(t_1) &\leq o(t_2). \end{aligned}$$

To show that $c(t_1) \leq c(t_2)$, assume the opposite, that $c(t_1) > c(t_2)$. Then, $t_1 < t_2 < c(t_2) < c(t_1) \leq t_1 + \mu(P)/2$, and $c(t_1) \in [t_2, t_2 + \mu(P)/2]$.

For a contradiction, suppose first that line segments $O(t_1)C(t_1)$ and $O(t_2)C(t_2)$ do not intersect, see Figure 7(a). Since $o(t_1) \leq o(t_2)$, the segment $O(t_2)C(t_2)$ lies completely under $O(t_1)C(t_1)$, and because $O(t_2)C(t_2)$ touches the plot in $C(t_2)$, the plot must intersect $O(t_1)C(t_1)$ in some point left of $C(t_2)$ and, hence, left of $C(t_1)$. That is a contradiction since $C(t_1)$ is the leftmost point where the lever for t_1 intersects the plot.

If line segments $O(t_1)C(t_1)$ and $O(t_2)C(t_2)$ do intersect, see Figure 7(b), then $C(t_1)$ lies under the line $O(t_2)C(t_2)$, and we we have

$$s(t_2, c(t_1)) < s(t_2, c(t_2)) = \min_{t \in [t_2, t_2 + \mu(P)/2]} s(t_2, t) \leq s(t_2, c(t_1)),$$

which again is a contradiction. This concludes the proof of the lemma. $\square$

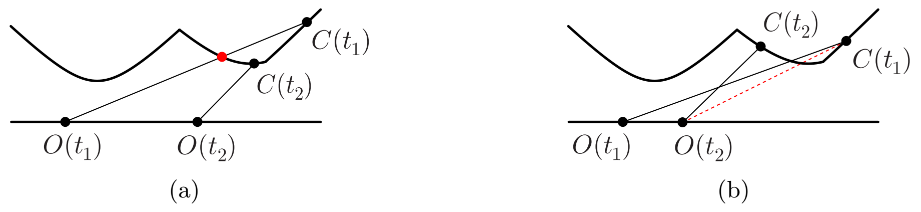


Figure 7: Two situations leading to a contradiction (the depicted points cannot be in this arrangement, resulting in invalid pictures).

4.2 States

In order to be able to simulate the continuous leftward motion of the lever, we transform it to an iteration over a discrete sequence of states. We define different lever states depending on how the lever is positioned relative to the sequence of hyperbola segments.

When $t \in [e_i, e_{i+1})$ and $c(t) \in [e_j, e_{j+1})$, we say that the lever for t is in the *phase* $\langle i, j \rangle$. The phase in which the lever is, together with the manner in which the lever touches the plot define the *state of the lever.* There are three possible ways for the lever to touch the plot, denoted by $\mathcal{K}$ (arc tangency), $\mathcal{Y}$ (endpoint sliding), and $\mathcal{V}$ (wedge touching).

- State $\langle i, j \rangle^{\mathcal{K}}$: $c(t) < t + \mu(P)/2$ and the lever is the tangent to $\mathcal{H}_j$.

  When t is decreasing, the lever is sliding to the left along h_j maintaining tangency, thus continuously decreasing its slope.

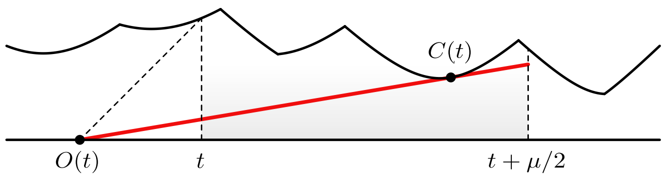


Figure 8: State $\langle i, j \rangle^{\mathcal{K}}$

- State $\langle i, j \rangle^{\mathcal{Y}}$: $c(t) = t + \mu(P)/2$.

  Point $C(t)$ is the right endpoint of the lever. It is the only point where the lever touches the plot. When t is decreasing, the lever is moving to the left while keeping its right endpoint on h_j.

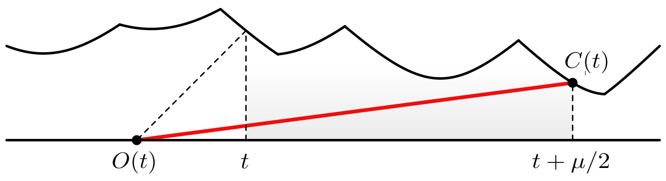


Figure 9: State $\langle i, j \rangle^{\mathcal{Y}}$

- State $\langle i, j \rangle^{\mathcal{V}}$: $c(t) < t + \mu(P)/2$ and the lever is passing through the point $H(e_j)$, the endpoint between hyperbola segments $\mathcal{H}_{j-1}$ and $\mathcal{H}_j$.

  This situation occurs only if $m_{j-1} > m_j$. The two neighboring hyperbola segments then form a "wedge" pointing downwards, and when t is decreasing the lever is sliding to the left while maintaining contact with the tip of the wedge, thus continuously decreasing its slope.

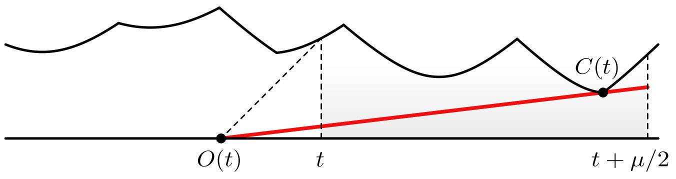


Figure 10: State $\langle i, j \rangle^{\mathcal{V}}$

4.3 Jumping and retargeting

In the process of decreasing t and dragging the left endpoint of the lever towards left, various events can take place. We will first devote some attention to the most challenging kind of events, which we call *jumping events*. These are events in which $C(t)$ abruptly changes its position by switching to a different hyperbola segment. In order to efficiently find state transition events that include jumps, we always need to know to which hyperbola segment we can jump from the current position. There is always at most one such target hyperbola segment, and we will show how to keep track of it.

Consider some point $H(x)$ on the plot. Let jump(x), *the jump destination for* x, be the index of the hyperbola segment which contains the rightmost point $H(w)$ on the plot such that $w < x$ and the ray from $H(x)$ through $H(w)$ only touches the plot, but does not intersect it properly, see Figure 11. That is, jump(x) is the index of the lowest visible hyperbola segment when looking from the point $H(x)$ to the left. If there is no such w, because hyperbola segments on the left are obscured by the segment containing $H(x)$, then we set jump(x) to be the index of the hyperbola segment containing $H(x)$.

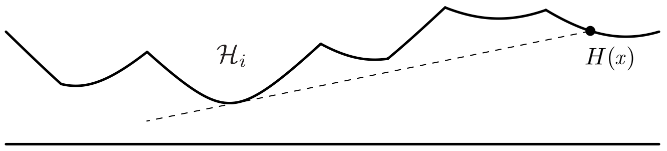
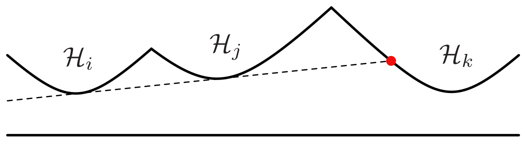


Figure 11: Jump destination for x. $jump(x) = i$

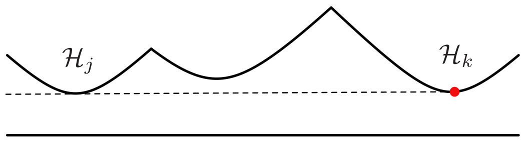
Consider only the values of x at which jump(x) changes value. We call such values *retargeting positions*, and points $H(x)$ *retargeting points.* There are two types of retargeting points, see Figure 12. *Retargeting points of the first type* are the points on $\mathcal{H}_k$ in which jump destination changes from i to j, where $i < j < k$, see Figure 12(a). *Retargeting points of the second type* are the points on $\mathcal{H}_k$ in which jump destination changes from k to j, where $j < k$, see Figure 12(b).

We construct an algorithm for finding all retargeting points. It is similar to finding lower convex chain in Andrew's monotone chain convex hull algorithm [9]. Our algorithm, however, runs in linear time because the sequence of hyperbola segments is already sorted. Before we give the algorithm, we describe the process and the supporting structure in more detail.

Given a set A of hyperbola segments, we take a look at the convex hull of their union



(a) Retargeting point of the first type.



(b) Retargeting point of the second type.

Figure 12: Two types of retargeting points.

and divide its boundary into the upper and lower part (i.e., the part lying above the segments, and the one below the segments). We are interested only in those hyperbola segments from A that have a nonempty intersection with the lower part of the convex hull boundary, having in mind that each hyperbola segment contains its left endpoint, but not the right one. Let us call the sequence of all such hyperbola segments, ordered from left to right, the *convex support* for A, see Figure 13.

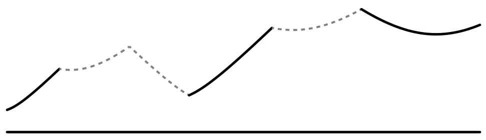


Figure 13: Convex support for all shown hyperbola segments is marked with solid lines.

We say that three hyperbola segments from the plot of h are in convex position if no line segment connecting a point from the left and a point from the right hyperbola segment passes completely below the middle hyperbola segment. Note that any three hyperbola segments of any convex support are in convex position.

Let j_0 be the index of a hyperbola segment that contains one of the global minima of h. Starting from $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}\}$, we process segments from left to right and maintain the convex support for the set $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \ldots, \mathcal{H}_k\}$, where k is the index of the segment being processed.

We use a stack to represent the convex support (only the indices of hyperbola segments are stored). Suppose the stack already contains the convex support for $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \ldots, \mathcal{H}_{k-1}\}$, and we want to add a new segment $\mathcal{H}_k$. We need to make changes to the stack so that it now represents the convex support for the new, extended, set $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \ldots, \mathcal{H}_k\}$. To achieve this, we pop segments from the stack until the last two segments still in the stack, together with $\mathcal{H}_k$, are in convex position. (Note that $\mathcal{H}_{j_0}$ will never be popped this way, as it contains a global minimum.) Finally, in the case where $\mathcal{H}_k$ belongs to the convex support of $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \ldots, \mathcal{H}_k\}$, we push it on the stack. (The intersection of $\mathcal{H}_k$ with the lower part of the convex hull is possibly empty since $\mathcal{H}_k$ does not contain its right endpoint.)

Let $\mathrm{cl}(X)$ denote the closure of a point set X, so $\mathrm{cl}(\mathcal{H}_i) = \mathcal{H}_i \cup \{E_{i+1}\}$, and let us consider the line l touching both $\mathrm{cl}(\mathcal{H}_i)$ and $\mathrm{cl}(\mathcal{H}_j)$, $i < j$, from below. If such a line is not unique, which can possibly happen only when $j = i + 1$, we take l to be the line with the smallest slope (that is, the line tangent to $\mathrm{cl}(\mathcal{H}_i)$ in E_{i+1}). We call the line l the *common tangent* of $\mathcal{H}_i$ and $\mathcal{H}_j$. It can be computed in a constant time, and in the following algorithm it is obtained as the return value of the function $\textsc{Tangent}(i, j)$.

Note that if Z is a point with larger first coordinate than the point $l \cap \mathrm{cl}(\mathcal{H}_j)$ (i.e., Z is to the right of $l \cap \mathrm{cl}(\mathcal{H}_j)$) and below the plot, then the point Z sees $\mathcal{H}_i$ lower than $\mathcal{H}_j$ if Z is below l, and $\mathcal{H}_j$ lower than $\mathcal{H}_i$ if Z is above l. We will use this fact in the analysis of the algorithm.

Now we are ready to present Algorithm 1, which shows how do we get retargeting points as intersections of hyperbola segments and common tangents of successive segments from the convex support.

Algorithm 1 Retargeting Points

```
Input: Description of a sequence of hyperbola segments.
Output: RetargetingPoints – the sequence of all retargeting points.
  RetargetingPoints ← [ ]
  Push($j_0 - 1$)
  Push($j_0$)
  for $k \leftarrow j_0 + 1$ to $j_0 + n$ do
      loop
          $i$ ← Second-to-top element of the stack
          $j$ ← Top element of the stack
          $l_1 \leftarrow$ Tangent($i, j$)
          if $l_1 \cap \mathcal{H}_k \neq \emptyset$ then
              Let $g$ be the leftmost point of $l_1 \cap \mathcal{H}_k$.
              Append $g$ to RetargetingPoints, and set jump destination of $g$ to $j$.
              Pop()
          else
              $l_2 \leftarrow$ Tangent($j, k$)
              if $l_2 \cap \mathcal{H}_k \neq \emptyset$ then
                  Let $g$ be the only point of $l_2 \cap \mathcal{H}_k$.
                  Append $g$ to RetargetingPoints, and set jump destination of $g$ to $j$.
                  Push($k$)
              end if
              break loop
          end if
      end loop
  end for
```

Theorem 1. *Algorithm 1 finds all retargeting positions, ordered from left to right, together with jump destinations of those positions, in $O(n)$ time.*

Proof. To show the correctness of this algorithm, we first observe that each reported point must be a retargeting point since the jump destination changes at it.

Indeed, points reported in the outer "if" branch lie on the common tangent of two successive hyperbola segments $\mathcal{H}_i$ and $\mathcal{H}_j$ from the convex support, and $\mathcal{H}_k$ is the first segment to be intersected by that tangent. The point of the intersection is the boundary between the region of $\mathcal{H}_k$ from which $\mathcal{H}_i$ is the lowest segment when looking to the left and the region of $\mathcal{H}_k$ for which such lowest segment is $\mathcal{H}_j$, as shown in Figure 12(a). Thus, a point g reported in this branch has the property that points on $\mathcal{H}_k$ just left and just right of the point g have $\mathcal{H}_j$ and $\mathcal{H}_i$ as their jump destinations, respectively, so it is a retargeting point of the first type.

The inner "if" branch occurs when the segment $\mathcal{H}_k$ is appended to the convex support, in which case there is a point on $\mathcal{H}_k$ acting as a boundary between the region of $\mathcal{H}_k$ from which no other segment is visible (when looking to the left), and the region of $\mathcal{H}_k$ from which $\mathcal{H}_j$ is visible, and such point lies on the common tangent of $\mathcal{H}_j$ and $\mathcal{H}_k$, as shown in Figure 12(b). Therefore, the point g reported in this branch has the property that points on $\mathcal{H}_k$ just left and just right of the point g have $\mathcal{H}_j$ and $\mathcal{H}_k$ as their jump destinations, respectively, so it is a retargeting point of the second type.

Next, let us make sure that no retargeting points were omitted by this algorithm. Consider a retargeting point g lying on $\mathcal{H}_k$.

If g is retargeting point of the first type, it must lie on $\textsc{Tangent}(i, j)$ for some $i < j < k$ (Figure 12(a)). Note that there can be no $\mathcal{H}_r$ with $r < k$ such that it reaches below $\textsc{Tangent}(i, j)$; otherwise r would be a jumping destination for g. That implies that both $\mathcal{H}_i$ and $\mathcal{H}_j$ are the part of the lower convex hull of the segments left from $\mathcal{H}_k$, which further means that $\mathcal{H}_i$ and $\mathcal{H}_j$ are two consecutive elements of the convex support for the set $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \dots, \mathcal{H}_{k-1}\}$. Since $\textsc{Tangent}(i, j)$ intersects $\mathcal{H}_k$, the same must be true for any pair of consecutive segments $\mathcal{H}_{i'}$ and $\mathcal{H}_{j'}$ from the convex support, with $i < i' < j' < k$. Otherwise, there would be three segments from the convex support not in convex position. The algorithm starts from the last two segments in the convex support and repeats moving to the previous pair as long as there is an intersection of the pair's common tangent with $\mathcal{H}_k$. That guarantees g will be found and reported as the retargeting point in the outer "if" branch.

The second case, when g is retargeting point of the second type, is treated similarly. In this case g lies on $\textsc{Tangent}(j, k)$ for some $j < k$ Figure 12(b). There can be no $\mathcal{H}_r$ with $r < k$ such that it reaches below $\textsc{Tangent}(j, k)$; otherwise r would be a jumping destination for g. That implies that $\mathcal{H}_j$ is a part of the lower convex hull of the segments left from $\mathcal{H}_k$, which further means that $\mathcal{H}_j$ is an element of the convex support for the set $\{\mathcal{H}_{j_0-1}, \mathcal{H}_{j_0}, \dots, \mathcal{H}_{k-1}\}$. Since $\textsc{Tangent}(j, k)$ touches $\mathcal{H}_k$ from below, the common tangent of each pair of consecutive segments $\mathcal{H}_{i'}$ and $\mathcal{H}_{j'}$, with $j \le i' < j' < k$, from the convex support must intersect $\mathcal{H}_k$. Otherwise, there would be three segments from the convex support not in convex position. For the same reason the common tangent of $\mathcal{H}_j$ and the segment immediately before it in the convex support cannot intersect $\mathcal{H}_k$. The algorithm starts from the last two segments in the convex support and repeats moving to the previous pair as long as there is an intersection of the pair's common tangent with $\mathcal{H}_k$. Finally, the algorithm reaches the rightmost pair of two consecutive segments from convex support whose common tangent does not intersect $\mathcal{H}_k$. The right segment from that pair is exactly $\mathcal{H}_j$. In that moment the point g is found and reported as retargeting point in the inner "if" branch.

The running time of algorithm is $O(n)$, since each index $k \in \{j_0 + 1, \dots, j_0 + n\}$ is pushed on the stack and popped from the stack at most once, and output of $\textsc{Tangent}()$ and intersections can be computed in a constant time. The number of retargeting points reported is, therefore, also $O(n)$.

Retargeting points reported by the algorithm come in sorted order, from left to right, which is explained by following observations. Retargeting points reported in a single iteration of the outer for-loop belong to the same hyperbola segment, and segments come in left-to-right order. Retargeting points reported on the same hyperbola segment are also in the same order: inside the inner loop, $\mathcal{H}_k$ is consecutively intersected with lines such that each line is of lower slope than previous and lies beneath it under $\mathcal{H}_k$. Hence, each subsequent intersection point lies to the right of the previous one.

This algorithm finds only retargeting points from a single period of the plotted function, but all other retargeting points can be obtained by simply translating these horizontally by the integral number of periods $\mu(P)$. □

4.4 Events

In the process of decreasing t and dragging the left endpoint of the lever towards left, the lever state changes at certain moments. We call such events *state transition events.* It is crucial for us to be able to efficiently calculate where these events can occur. If the current lever position, t_c, and the current state are known, the following event can be determined by maintaining the set of conceivable future events of which at least one must be realized, and proceeding to the one that is the first to take place, i.e., the one with the largest t not larger than t_c. To do that, we must know how to calculate the value of t for each event.

We list all types of events that can happen while moving the lever leftwards, and for each we show how to calculate the value of t, the lever position at which the event occurs. We will give a polynomial equation describing each event type, which will be solved either for t or for $o_i(t)$. Once we have $o_i(t)$, it is easy to obtain t, as

$$t = \frac{o_i(t)^2 - d_i^2 - m_i^2}{2(o_i(t) - m_i)}. \tag{4}$$

In the process of determining t, we will repeatedly encounter fixed degree polynomial equations. Solving them can be assumed to be a constant time operation, see [10].

We will also frequently use the following two utility functions, $c_j(o)$ and $s_j(o)$.

For $o < m_j$, let $c_j(o)$ be such that $H(c_j(o)) := (c_j(o), h(c_j(o)))$ is the contact point of the hyperbola h_j and its tangent through the point $(o, 0)$. Given o, the value $c_j(o)$ can be calculated by solving the equation $h_j'(c_j(o)) = h_j(c_j(o))/(c_j(o) - o)$, which results in

$$c_j(o) = \frac{d_j^2 + m_j^2 - m_j o}{m_j - o}. \tag{5}$$

Function $s_j(o)$ is defined as the slope of the tangent to the hyperbola h_j through the point $(o, 0)$, that is, the line through points $(o, 0)$ and $H(c_j(o))$,

$$s_j(o) = \frac{h_j(c_j(o))}{c_j(o) - o} = 1/\sqrt{\left(\frac{m_j - o}{d_j}\right)^2 + 1}. \tag{6}$$

These functions are used when we know that the lever for t is a tangent to some hyperbola segment $\mathcal{H}_j$. Then we know that the lever touches $\mathcal{H}_j$ at the point with coordinate $c_j(o(t))$, and that its slope equals $s_j(o(t))$.

Let us first consider *jump destination change event*, a type of event which is not a state transition event. Nevertheless, it is still necessary to react to events of this kind in order to update a parameter needed for calculating events that do change state. As we decrease t, we keep track of jump destination for position $c(t)$ in variable $j_m := \text{jump}(c(t))$.

4.4.1 Jump destination change event

Jump destination j_m changes whenever $C(t)$ passes over some retargeting point. At that moment it is necessary to recalculate all future events involving jumps, since j_m is used for their calculation. Let z be the next retargeting position, i.e., the rightmost one that lies to the left of $c(t_c)$, where t_c is current lever position. Depending on the lever state, we calculate the event position in one of the following ways.

- The current state is $\langle i,j\rangle^{\mathcal{V}}$

  The jump destination change event cannot occur before leaving this state since $C(t)$ stands still at the "wedge tip", so it cannot pass over any retargeting point.

- The current state is $\langle i,j\rangle^{\mathcal{K}}$

  Here the lever is tangent to $\mathcal{H}_j$, so this event can only happen if $z > m_j$. Otherwise, the lever would have nonpositive slope when touching the plot at $H(z)$. The equation describing this event is
  $$c_j(o_i(t)) = z,$$
  and it solves to
  $$o_i(t) = \frac{d_j^2 - zm_j + m_j^2}{m_j - z}.$$

- The current state is $\langle i,j\rangle^{\mathcal{Y}}$

  The right endpoint of the lever slides over $\mathcal{H}_j$ and will coincide with $H(z)$ at
  $$t = z - \mu(P)/2.$$

Note that j_m is not used in the description of the lever state, so, as already noted, jump destination change event does not change the current state.

All the other events that need to be considered are state transition events.

4.4.2 State transition events

In the following list we give all possible types of state transition events, and we show how to calculate corresponding t value for each of them. To make this enumeration easier to follow, we group event types in four groups, depending on the resulting phase of the event: $\langle i,j\rangle$, $\langle i-1,j\rangle$, $\langle i,j-1\rangle$ and $\langle i,j_m\rangle$.

Event types leading to $\langle i,j\rangle$ phase:

- $\langle i,j\rangle^{\mathcal{Y}} \to \langle i,j\rangle^{\mathcal{K}}$ and $\langle i,j\rangle^{\mathcal{K}} \to \langle i,j\rangle^{\mathcal{Y}}$

  In this event the lever changes from being a tangent to $\mathcal{H}_j$ to touching $\mathcal{H}_j$ with its right endpoint, or the other way round. The corresponding equation for this event is
  $$c_j(o_i(t)) = t + \mu(P)/2,$$
  which can be transformed into a cubic equation in t.

Since there can be at most three real solutions to that equation, it is possible that this event takes place at most three times for same i and j. On each occurrence of the event the lever switches between being a tangent and touching the plot with its right endpoint.

- $\langle i,j\rangle^{\mathcal{K}} \to \langle i,j\rangle^{\mathcal{V}}$

  This event happens when the point in which the lever is touching $\mathcal{H}_j$ reaches e_j. Here, the lever is tangent to $\mathcal{H}_j$, and since it must have a positive slope, this will only happen if $e_j > m_j$. The equation for the event is

$$c_j(o_i(t)) = e_j,$$

  which solves to

$$o_i(t) = \frac{d_j^2 - e_j m_j + m_j^2}{m_j - e_j}.$$

Event types leading to $\langle i-1,j\rangle$ phase:

- $\langle i,j\rangle^{x} \to \langle i-1,j\rangle^{x}$, where $x \in \{\mathcal{Y},\mathcal{K},\mathcal{V}\}$

  This is the event when the interval to which t belongs changes from $[e_i, e_{i+1})$ to $[e_{i-1}, e_i)$, so this event happens at e_i,

$$t = e_i.$$

Event types leading to $\langle i,j-1\rangle$ phase:

- $\langle i,j\rangle^{\mathcal{Y}} \to \langle i,j-1\rangle^{\mathcal{Y}}$

  Here, the right endpoint of the lever slides continuously from one hyperbola segment to another,

$$t = e_j - \mu(P)/2.$$

- $\langle i,j\rangle^{\mathcal{V}} \to \langle i,j-1\rangle^{\mathcal{Y}}$

  This event happens when the lever stops touching the tip of the wedge and starts to slide its right endpoint over the hyperbola segment on the left of the wedge,

$$t = e_j - \mu(P)/2.$$

- $\langle i,j\rangle^{\mathcal{V}} \to \langle i,j-1\rangle^{\mathcal{K}}$

  This event happens when the lever stops touching the tip of the wedge and becomes a tangent of the hyperbola segment on the left of the wedge. This can only happen if $e_j > m_{j-1}$,

$$c_{j-1}(o_i(t)) = e_j,$$

  which solves to

$$o_i(t) = \frac{d_{j-1}^2 - e_j m_{j-1} + m_{j-1}^2}{m_{j-1} - e_j}.$$

Event types leading to $\langle i, j_m\rangle$ phase:

- $\langle i, j\rangle^{\mathcal{Y}} \to \langle i, j_m\rangle^{\mathcal{K}}$

  This event happens when the lever state changes from having an endpoint on $\mathcal{H}_j$ to being a tangent to $\mathcal{H}_{j_m}$. The corresponding equation is

  $$s_{j_m}(o_i(t)) = \frac{h_j(t + \mu(P)/2)}{h_i(t) + \mu(P)/2},$$

  which further transforms into a polynomial equation in t.

  The line through $o_i(t)$ with slope $s_{j_m}(o_i(t))$ touches the hyperbola h_{j_m}, but we need to be sure that it actually touches the segment $\mathcal{H}_{j_m}$ of that hyperbola. It may as well be the case that $\mathcal{H}_{j_m}$ is not wide enough to have a common point with the line. More precisely, the first coordinate, u, of the touching point between the line and h_{j_m} must belong to the interval $[e_{j_m}, e_{j_m+1})$. To get that coordinate, we solve the equation

  $$h'_{j_m}(u) = s_{j_m}(o_i(t)).$$

  Having in mind that $o_i(t) < m_{j_m} < u$ must hold, we get a single solution

  $$u = m_{j_m} + \frac{d_{j_m}^2}{m_{j_m} - o_i(t)}.$$

  If $u \notin [e_{j_m}, e_{j_m+1})$, we do not consider this event.

  Checking if the line through $o_i(t)$ with slope $s_{j_m}(o_i(t))$ actually touches the hyperbola segment $\mathcal{H}_{j_m}$ will also be used in the calculation for the following event types, where we will refer to it by the name *collision check*.

- $\langle i, j\rangle^{\mathcal{K}} \to \langle i, j_m\rangle^{\mathcal{K}}$

  This event happens when the lever becomes a tangent to two hyperbola segments, $\mathcal{H}_j$ and $\mathcal{H}_{j_m}$ simultaneously. It can only happen if $\mathcal{H}_{j_m}$ is lower than $\mathcal{H}_j$, i.e., $d_{j_m} < d_j$,

  $$s_{j_m}(o_i(t)) = s_j(o_i(t)).$$

  Since $o_i(t) < m_{j_m}$ and $o_i(t) < m_j$, the only solution is

  $$o_i(t) = \frac{d_j m_{j_m} - d_{j_m} m_j}{d_j - d_{j_m}}.$$

  Here we need to apply the collision check described earlier to see if the common tangent actually touches $\mathcal{H}_{j_m}$. If the check fails, we do not consider this event.

- $\langle i, j\rangle^{\mathcal{V}} \to \langle i, j_m\rangle^{\mathcal{K}}$

  This event happens when the lever stops touching the tip of the wedge and becomes a tangent of the hyperbola segment $\mathcal{H}_{j_m}$,

  $$s_{j_m}(o_i(t)) = \frac{h_j(e_j)}{e_j - o_i(t)}.$$

  This can only happen if $h_j(e_j) > d_{j_m}$. From that we get a quadratic equation in $o_i(t)$. The two solutions correspond to the two tangents to h_{j_m} from the point $(e_j, h(e_j))$, so we consider only the smaller solution, which corresponds to the left tangent. Once again we perform the collision check to see if the tangent actually touches $\mathcal{H}_{j_m}$, otherwise we disregard this event.

- $\langle i, j\rangle^{\mathcal{Y}} \to \langle i, j_m\rangle^{\mathcal{V}}$

  The event when the lever state changes from having an endpoint on $\mathcal{H}_j$ to touching the wedge between $\mathcal{H}_{j_{m-1}}$ and $\mathcal{H}_{j_m}$ is described by

$$\frac{h_{j_m}(e_{j_m})}{e_{j_m} - o_i(t)} = \frac{h_j(t + \mu(P)/2)}{h_i(t) + \mu(P)/2},$$

  which again transforms into a polynomial equation in t.

- $\langle i, j\rangle^{\mathcal{K}} \to \langle i, j_m\rangle^{\mathcal{V}}$

  The event in which the lever touches the wedge tip at e_{j_m} while being a tangent to h_j is represented by the following equation,

$$\frac{h_{j_m}(e_{j_m})}{e_{j_m} - o_i(t)} = s_j(o_i(t)).$$

  This can only happen if $h_{j_m}(e_{j_m}) < d_j$. It can be transformed into a quadratic equation in $o_i(t)$. The two solutions correspond to the two tangents to h_j from the point $(e_{j_m}, h(e_{j_m}))$. The smaller of the two solutions is where this event happens.

- $\langle i, j\rangle^{\mathcal{V}} \to \langle i, j_m\rangle^{\mathcal{V}}$

  This event happens when the lever touches two wedges, at points E_{j_m} and E_j simultaneously. This condition can be written as

$$\frac{h_j(e_j)}{e_j - o_i(t)} = \frac{h_{j_m}(e_{j_m})}{e_{j_m} - o_i(t)},$$

  which is a linear equation in $o_i(t)$.

4.5 Sequence of realized states

We want to efficiently find the sequence of states through which the lever will pass on its leftward journey, together with the positions where these state changes happen. Let the obtained sequence be $p_1, \mathcal{S}_1, p_2, \mathcal{S}_2, p_3, \ldots, p_r, \mathcal{S}_r$, where $p_1 \leq p_2 \leq \ldots \leq p_r$. Each state $\mathcal{S}_k$ occurs when the lever position is exactly between p_k and p_{k+1}, where $p_{r+1} = p_1 + \mu(P)$. We call this sequence *the sequence of realized states.*

To calculate the sequence of realized states, we will start from a specific lever position that has a known state. Let p_{low} be any of the values for which h attains its global minimum, and let $\mathcal{H}_{j_0}$ be the hyperbola segment above it. The algorithm starts with the lever in the position $t_c = t_0 = p_{low} - \mu(P)/2$. This lever has its right endpoint on the plot at the point $H(p_{low})$, which means that its state is $\langle i_0, j_0\rangle^{\mathcal{Y}}$, where i_0 is the index of the hyperbola segment over t_0. We note that p_{low} must also be a retargeting position, so we also know our initial jump destination.

The algorithm then iterates with the following operations in its main loop. It first calculates all possible events that could happen while in the current state. Among those events let E be the one with the largest t that is not larger than t_c. It is the event that must occur next. The algorithm sets t_c to t, and it either updates jump destination if E is a jump destination change event, or switches to the new state if the event is a state transition event. In the latter case, the position t and the new state are added to the sequence of realized states. These operations are repeated until one full period of the plot is swept, ending with $t_c = t_0 - \mu(P)$ in the state $\langle i_0 - n, j_0 - n\rangle^{\mathcal{Y}}$. The procedure described is given in Algorithm 2.

Algorithm 2 Sliding Lever Algorithm

```
Input: Description of a sequence of hyperbola segments.
Output: The sequence of realized states.
  Find $p_{low}$ and $j_0$.
  $t_0 \leftarrow p_{low} - \mu(P)/2$
  Find $i_0$.
  Run Retargeting Points to find retargeting points and their jump destinations.
  $t_c \leftarrow t_0$
  $i \leftarrow i_0$
  $j \leftarrow j_0$
  $j_m \leftarrow$ jump destination of $p_{low}$
  Set the current state to $\langle i_0, j_0\rangle^{\mathcal{Y}}$.
  Add $t_0$ and $\langle i_0, j_0\rangle^{\mathcal{Y}}$ to the output sequence.
  while $t_c > t_0 - \mu(P)$ do
      Calculate all the events for the current state. Ignore jumping events if $j = j_m$.
      Let $E$ be the first event to happen (the one with the largest $t$ not larger than $t_c$).
      $t_c \leftarrow t$ of the event $E$.
      if $E$ is jump destination change event then
          Update $j_m$.
      else
          Set the current state to the destination state of $E$.
          Add $t$ and the current state to the output sequence.
          if $E$ is a jumping event then
              $j_m \leftarrow j$
          end if
      end if
  end while
```

Theorem 2. Sliding Lever Algorithm *finds the sequence of realized states in $O(n)$ time. The length of the produced sequence is $O(n)$.*

Proof. During the execution of the algorithm we must encounter all realized states, since states can change only at events and by always choosing the first following possible event to happen, we eventually consider all realized events. Realized states are encountered in order, since t_c is never increasing.

While choosing the following event we did not consider the possibility that there can be several events with the same minimal t. However, if that happens we can choose an arbitrary one to be the next event. This choice can influence the output sequence only by including or excluding some states of the length zero. Importantly, such zero-length states are irrelevant for further considerations, and no other state in the output of the algorithm is influenced by this choice.

Each event is either a jump destination change event or a state transition event. From

Theorem 1 we have that there are $O(n)$ jump destination change events, and now we will show that there are $O(n)$ state transition events as well.

Each state transition event transitioning from some $\langle i,j\rangle$ state decreases either i, or j or both. The only exception are the events $\langle i,j\rangle^{\mathcal{K}} \to \langle i,j\rangle^{\mathcal{Y}}$ and $\langle i,j\rangle^{\mathcal{Y}} \to \langle i,j\rangle^{\mathcal{K}}$, however those events can happen at most three times in total for the same i and j. Note that $j_m \leq j$, but when $j_m = j$, we do not consider events involving j_m. Variables i and j start with values i_0 and j_0, and, after the loop finishes, they are decreased to $i_0 - n$ and $j_0 - n$. Hence, no more than $O(n)$ state transition events occurred, implying the linear length of the sequence of realized states.

Calculation of each state transition event takes a constant time, at each iteration a constant number of potential events is considered, and loop is iterated $O(n)$ times. To find the next jump destination change event, we move through the sorted list of retargeting positions until we encounter the first retargeting position not greater than t_c. The total time for calculating jump destination change events, over all iterations, is linear. Therefore, the running time of the whole algorithm is also linear. $\square$

5 Merging the two dilations

In this section we will show how to use output of the sliding lever algorithm to give an answer to our original question. The algorithm is run once for the left and once the right dilation, and the obtained sequences of realized states are merged into a single sequence. Finally, we will explain how to find the overall minimum dilation and the optimal feed-link by iterating through the merged sequence.

Knowing the sequence of realized states is sufficient to determine the exact lever slope at any position. Remember, the lever slope at position t is the inverse of the left dilation via $P(t)$, as shown in (3). But, to know the dilation via some point we need both the left and the right dilations via that point (1).

Our sliding lever algorithm was initially designed only for the left dilation, but an analogous algorithm can obviously be designed for the right dilation (or, we can perform the *exact* same algorithm for the left dilation on the mirror image of the polygon P, and then transform obtained results appropriately). This implies the concept of the right dilation lever for t (as opposed to the left dilation lever, or just the lever, as we have been calling it until now), which has negative slope and touches the plot on the left side of t. We will use $+$ and $-$ in superscript denoting relation with the left and the right dilation, respectively.

Let $p_1^+, \mathcal{S}_1^+, p_2^+, \mathcal{S}_2^+, p_3^+, \ldots, p_{r^+}^+, \mathcal{S}_{r^+}^+$ and $p_1^-, \mathcal{S}_1^-, p_2^-, \mathcal{S}_2^-, p_3^-, \ldots, p_{r^-}^-, \mathcal{S}_{r^-}^-$ be the sequences of realized states for the left and the right dilation, respectively, where both sequences p^+ and p^- are in nondecreasing order. For simplicity, let us call them the *left* and the *right* sequence, respectively. States for right dilation are described by $\langle i,j\rangle$ notation as well, with the meaning analogous to the meaning of the notation for the left dilation states. We say that the (right dilation) lever for t is in the phase $\langle i,j\rangle$, when $\mathcal{H}_i$ is the hyperbola segment above t, and the (right dilation) lever touches the hyperbola segment $\mathcal{H}_j$.

We now merge the two obtained sequences by overlapping them into a new sequence $p_1, \mathcal{S}_1, p_2, \mathcal{S}_2, p_3, \ldots, p_r, \mathcal{S}_r$. In the merged sequence, $p_1 \leq p_2 \leq \ldots \leq p_r$ is the sorted union of $\{p_1^+, p_2^+, \ldots, p_{r^+}^+\}$ and $\{p_1^-, p_2^-, \ldots, p_{r^-}^-\}$. States in the merged sequence are pairs consisting of one state from the left sequence and one state from the right sequence. Each state $S_k = (\mathcal{S}_{k^+}^+, \mathcal{S}_{k^-}^-)$ in the merged sequence is such that $\mathcal{S}_{k^+}^+$ and $\mathcal{S}_{k^-}^-$ are states covering the interval between p_k and p_{k+1} in the left and in the right sequence, respectively.

By Theorem 2, both r^+ and r^- are $O(n)$, so the length of the merged sequence is also linear in n. Because the left and the right sequences are sorted, the merged sequence can be computed in $O(n)$ time.

For each state $\mathcal{S}_k = (\mathcal{S}_{k^+}^+, \mathcal{S}_{k^-}^-)$ there is a single expression for computing the lever slope as a function of t, when $p_k \le t \le p_{k+1}$, both for the left and the right dilation. To find the minimal dilation while in that state, we want to find t which maximizes the minimum of the two slopes for the left and the right lever. This observation readily follows from (1) and (3), so

$$\min_{p_k \le t \le p_{k+1}} \tilde{\delta}_{P(t)} = \frac{1}{\max_{p_k \le t \le p_{k+1}} \min\{\tilde{s}^+(t), \tilde{s}^-(t)\}}, \tag{7}$$

where $\tilde{s}^+(t)$ is the slope for the left dilation lever for t, and $\tilde{s}^-(t)$ is the slope for the right dilation lever for t.

This means that to get the final value of the optimal dilation we essentially need to compute the minimum of the upper envelope of two functions, which is a standard procedure that can be done efficiently. In the following, we take a closer look at the computations needed to complete this step in our setting.

For all possible combinations of the left and the right state types in a combined state, we show how to find the maximum of the lower envelope of the slope functions $\tilde{s}^+(t)$ and $\tilde{s}^-(t)$ by analyzing the shape of those functions. The following lemmas help us describe them.

Let us assume that the corresponding state to which t belongs is $\mathcal{S} = (\mathcal{S}^+, \mathcal{S}^-)$, and let $s^+_{\langle i,j\rangle^{\mathcal{K}}}(t)$ be a function which maps t to the slope of a lever, assuming that the lever is in $\langle i,j\rangle^{\mathcal{K}}$ state. Analogously we define $s^+_{\langle i,j\rangle^{\mathcal{V}}}(t)$, $s^+_{\langle i,j\rangle^{\mathcal{Y}}}(t)$, $s^-_{\langle i,j\rangle^{\mathcal{K}}}(t)$, $s^-_{\langle i,j\rangle^{\mathcal{V}}}(t)$ and $s^-_{\langle i,j\rangle^{\mathcal{Y}}}(t)$.

Lemma 3. *If $\mathcal{S}^+$ is a $\langle i,j\rangle^{\mathcal{K}}$ state, then $\tilde{s}^+(t)$ is a monotonically nondecreasing function.*

Proof. If $\mathcal{S}^+$ is an $\langle i,j\rangle^{\mathcal{K}}$ state, then, from equation (6), we have

$$\tilde{s}^+(t) = s^+_{\langle i,j\rangle^{\mathcal{K}}}(t) = s_j(o_i(t)) = \frac{h_j(c_j(o_i(t)))}{c_j(o_i(t)) - o_i(t)} = 1/\sqrt{\left(\frac{m_j - o_i(t)}{d_j}\right)^2 + 1}.$$

We see that the function s_j is monotonically increasing for parameter values less than m_j. In the specified state, $o_i(t) < m_j$ holds, and since $o(t)$ is monotonically nondecreasing (Lemma 2), it means that $\tilde{s}^+(t)$, being the composition of s_j and $o(t)$, is monotonically nondecreasing as well. □

Lemma 4. *If $\mathcal{S}^+$ is a $\langle i,j\rangle^{\mathcal{V}}$ state, then $\tilde{s}^+(t)$ is a monotonically nondecreasing function.*

Proof. If $\mathcal{S}^+$ is an $\langle i, j\rangle^{\mathcal{V}}$ state, then we have

$$\tilde{s}^+(t) = s^+_{\langle i,j\rangle^{\mathcal{V}}}(t) = \frac{h_j(e_j)}{e_j - o_i(t)}.$$

We see that $\tilde{s}^+(t)$ is monotonically increasing in terms of $o_i(t)$ when $o_i(t) < e_j$, which holds in the specified state. Since $o(t)$ is monotonically nondecreasing (Lemma 2), it means that $\tilde{s}^+(t)$ is monotonically nondecreasing in terms of t as well. □

Similar observations hold for the right dilation analogues: $\tilde{s}^-(t)$ is monotonically nonincreasing if $\mathcal{S}^-(t)$ is $\langle i, j\rangle^{\mathcal{K}}$ or $\langle i, j\rangle^{\mathcal{V}}$ state.

Lemma 5. *If $\mathcal{S}^-$ is a $\langle i, j\rangle^{\mathcal{Y}}$ state then $\tilde{s}^+(t) \leq \tilde{s}^-(t)$.*

Proof. From equation (3), using the fact that $h_j(t) = h_{j+n}(t + \mu(P))$ holds because of the periodicity, we have

$$\begin{aligned}\tilde{s}^-(t) &= \frac{h_j(t - \mu(P)/2)}{h_i(t) + \mu(P)/2}\\ &= \frac{h_{j+n}(t + \mu(P)/2)}{h_i(t) + \mu(P)/2}\\ &= s(t, t + \mu(P)/2)\\ &\geq \min_{t_r \in [t, t+\mu(P)/2]} s(t, t_r)\\ &= \tilde{s}^+(t).\end{aligned}$$

□

Analogously, if $\mathcal{S}^+$ is a $\langle i, j\rangle^{\mathcal{Y}}$ state then $\tilde{s}^-(t) \leq \tilde{s}^+(t)$.

For equation (7) we need to calculate $\max_{p_k \leq t \leq p_{k+1}} \min\{\tilde{s}^+(t), \tilde{s}^-(t)\}$. This calculation depends on the types of the states $\mathcal{S}^+$ and $\mathcal{S}^-$ in the specified interval. We analyze nine possible type combinations.

- If $\mathcal{S}^+$ is $\langle i, j^+\rangle^{\mathcal{Y}}$ state and $\mathcal{S}^-$ is $\langle i, j^-\rangle^{\mathcal{Y}}$ state:

  Using Lemma 5 we get $\tilde{s}^+(t) = \tilde{s}^-(t)$, so

  $$\max_{p_k \leq t \leq p_{k+1}} \min\{\tilde{s}^+(t), \tilde{s}^-(t)\} = \max_{p_k \leq t \leq p_{k+1}} s^+_{\langle i,j^+\rangle^{\mathcal{Y}}}(t).$$

  The maximum is achieved either at interval ends or at a local maxima, if one exists, which is found by solving a polynomial equation.

- If $\mathcal{S}^+$ is $\langle i, j^+\rangle^{\mathcal{K}}$ state and $\mathcal{S}^-$ is $\langle i, j^-\rangle^{\mathcal{Y}}$ state:

  Using Lemma 5 and Lemma 3 we get

  $$\max_{p_k \leq t \leq p_{k+1}} \min\{\tilde{s}^+(t), \tilde{s}^-(t)\} = \max_{p_k \leq t \leq p_{k+1}} s^+_{\langle i,j^+\rangle^{\mathcal{K}}}(t) = s^+_{\langle i,j^+\rangle^{\mathcal{K}}}(p_{k+1}).$$

- If $\mathcal{S}^+$ is $\langle i, j^+\rangle^{\mathcal{K}}$ state and $\mathcal{S}^-$ is $\langle i, j^-\rangle^{\mathcal{K}}$ state:

  From Lemma 3 we know that $s^+_{\langle i,j^+\rangle^{\mathcal{K}}}(t)$ is monotonically nondecreasing, and $s^-_{\langle i,j^-\rangle^{\mathcal{K}}}(t)$ is monotonically nonincreasing. The highest point of the lower envelope of these functions on $[p_k, p_{k+1}]$ is thus located either at one end of the interval, or at the point of the intersection of the two functions, which can be found by solving a polynomial equation.

The other six combinations of state types are not listed, but each of them is resolved in a manner very similar to the one of the above three combinations. Cases with $\langle\cdot,\cdot\rangle^{\mathcal{V}}$ states are resolved analogously to the cases having $\langle\cdot,\cdot\rangle^{\mathcal{K}}$ states instead by using Lemma 4 in place of Lemma 3, and the remaining cases are analogous to the cases having "pluses" and "minuses" swapped.

Finally, by taking the smallest of all dilation minima from $[p_k, p_{k+1}]$ intervals for $k \in \{1, 2, \ldots r\}$ we obtain the overall minimum dilation,

$$\delta = \min_{k\in\{1,2,\ldots,r\}} \min_{p_k\le t\le p_{k+1}} \tilde{\delta}_{P(t)}.$$

While going through calculated interval minima we maintain the value of t for which the minimum is achieved, so we also get the point on P which is the endpoint of the optimal feed-link.

6 Conclusion

The problem we considered asked for the optimal extension of a polygonal network by connecting a specified point to the rest of the network via a feed-link. We gave a linear time algorithm for solving this problem, thus improving upon the previously best known result of Aronov et al. presented in [3].

On the way to solution, we performed several steps. First, we divided the concept of dilation into the left and the right dilation, so that the two can be analyzed separately. Then we transformed them into the problem which considers the plot of the distance function and lever slopes, an gave an algorithm for event based simulation of the lever movement. The output of the algorithm is the description of the changes in the lever slope presented as a sequence of states, each of which can be expressed analytically. Finally, we explained how those state sequences for the left and the right dilation can be merged and how the optimal feed-link can be found from it.

We note that the method we used for solving the original problem can be easily adapted to work with any network shaped as an open polygonal chain.

Aronov et al. in [3] discuss polygons with obstacles. They show how polygonal obstacles with total number of vertices equal to b induce a partition of the polygon boundary of the size $O(nb)$, and how that partition can be computed in $O(nb + b\log b)$ time. Each segment of the partition has a distance function to p similar to function $h_i(t)$, the only difference being an additive constant. In our setting, this makes the hyperbola segments in the plot to shift upward by that constant, which means that the problem with obstacles can also be handled by our approach. This variation would require a more elaborate case study from Section 4.4.2. The equations in respective cases would also change, but they would still remain polynomial and thus efficiently solvable. We will not pursue the details in the present paper.

A natural way to generalize the problem is to assume that the polygon edges are not line segments, but curves (second order curves, for example). The abstraction behind our method can be applied in this case provided that there is an efficient way to determine the event times and to find optimal values in the merged sequence.

Another obvious generalization is asking for a set of $k > 1$ feed-links that minimize the dilation. Aronov et al. in [3] give an approximate algorithm for computing such a set. However, finding an efficient algorithm that solves this problem exactly is still open. One could try to apply our approach here, but we do not see that this can be done in a straightforward fashion.

It would also be interesting to see whether a similar method can be applied to a network which is not necessarily polygonal, i.e., to a network whose vertices can have degree greater than two.

Acknowledgments

We would like to thank the anonymous referees whose comments and insights improved our paper.

References

[1] A. Dahlgren and L. Harrie. Evaluation of computational methods for connecting points to large networks. *Mapping and Image Science*, (4):45–54, 2006.

[2] A. Dahlgren and L. Harrie. Development of a tool for proximity applications. In *Proceedings of AGILE*, 2007.

[3] B. Aronov, K. Buchin, M. Buchin, B. Jansen, T. de Jong, M. van Kreveld, M. Löffler, J. Luo, R.I. Silveira, and B. Speckmann. Connect the dot: Computing feed-links for network extension. *Journal of Spatial Information Science*, (3):3–31, 2012.

[4] P. Bose, K. Dannies, J.L. De Carufel, C. Doell, C. Grimm, A. Maheshwari, S. Schirra, and M. Smid. Network farthest-point diagrams and their application to feed-link network extension. *Journal of Computational Geometry*, 4(1):182–211, 2013.

[5] G. Narasimhan and M. Smid. Approximating the stretch factor of euclidean graphs. *SIAM Journal on Computing*, 30(3):978–989, 2000.

[6] A. Ebbers-Baumann, R. Klein, E. Langetepe, and A. Lingas. A fast algorithm for approximating the detour of a polygonal chain. *Computational Geometry Theory and Applications*, 27(2):123–134, 2004.

[7] P.K. Agarwal, R. Klein, C. Knauer, S. Langerman, P. Morin, and M. Soss. Computing the detour and spanning ratio of paths, trees and cycles in 2d and 3d. *Discrete and Computational Geometry*, 39(1–3):17–37, 2008.

[8] A. Ebbers-Baumann, A. Grune, and R. Klein. On the geometric dilation of finite point sets. *Algorithmica*, 44(2):137–149, 2006.

[9] A.M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979.

[10] A. Dickenstein and I.Z. Emiris. *Solving polynomial equations: Foundations, algorithms, and applications*, volume 14. Springer Verlag, 2005.