International Journal of Computational Geometry & Applications

APPROXIMATING POLYGONS AND SUBDIVISIONS WITH MINIMUM-LINK PATHS

LEONIDAS J. GUIBAS
Dept. of Computer Science, Stanford, CA USA 94305
DEC SRC, 130 Lytton Ave, Palo Alto, CA USA 94301

JOHN E. HERSHBERGER
DEC SRC, 130 Lytton Ave, Palo Alto, CA USA 94301

JOSEPH S. B. MITCHELL*
Dept. of Applied Math, SUNY, Stony Brook, NY USA 11794–3600

JACK SCOTT SNOEYINK†
Dept. of Computer Science, UBC, Vancouver, B.C. Canada V6T 1Z2

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

We study several variations on one basic approach to the task of simplifying a plane polygon or subdivision: Fatten the given object and construct an approximation inside the fattened region. We investigate fattening by convolving the segments or vertices with disks and attempt to approximate objects with the minimum number of line segments, or with near the minimum, by using efficient greedy algorithms. We give some variants that have linear or $O(n \log n)$ algorithms approximating polygonal chains of n segments. We also show that approximating subdivisions and approximating with chains with no self-intersections are *NP*-hard.

Keywords: Polygonal approximation, link metric, cartographic line simplification, curve segmentation, Fréchet metric

1. Introduction

In the practical application of computers to graphics, image processing, and geographic information systems, great gains can be made by replacing complex geometric objects with simpler objects that capture the relevant features of the original. The need for simplification is most clearly seen in cartography. McMaster[29]

*Partially supported by a grant from Hughes Research Laboratories, Malibu, CA, and by NSF Grant ECSE-8857642.

†Portions of this research were performed while visiting Utrecht University and being supported by the ESPRIT Basic Research Action No. 3075 (project ALCOM).

lists ways that current methods and technology benefit from data simplification and reduction, including reduced storage space and faster vector operations, vector to raster conversion, and plotting. Improving computation and plotting capabilities does not always help; currently, the speed of data communication is often the bottleneck. Even manual cartography depends on simplification: boundaries must be simplified when drawing a map at a smaller scale or the map becomes unreadable because of the inconsequential information it presents. A good example is the map in Lewis Carroll's *Sylvia and Bruno* with a scale of 1:1.

The theme of our approach to the task of simplifying a plane path, polygon, or subdivision is: *Fatten the given object and construct an approximation inside the fattened region.* This theme has many variations. In this section, we consider some variants that apply to the cartographers' *line simplification problem.* In section 2 we briefly survey the literature on this and related approximation problems.

A list of n points $p_1, p_2, \ldots, p_n$ defines a *polygonal chain* with line segments or *links* $\overline{p_i p}_{i+1}$. Given a polygonal chain C, the line simplification problem asks for a polygonal chain $\widetilde{C}$ with fewer than n links that represents C well. If the criterion of representing C well is that every point of the approximation $\widetilde{C}$ be within ε of a point of C, then the following fattening method could be used. Paint C with a circular brush of radius ε to obtain a fattened region. Then use a minimum-link path algorithm to approximate C within the fattened region, as illustrated in figure 1a.

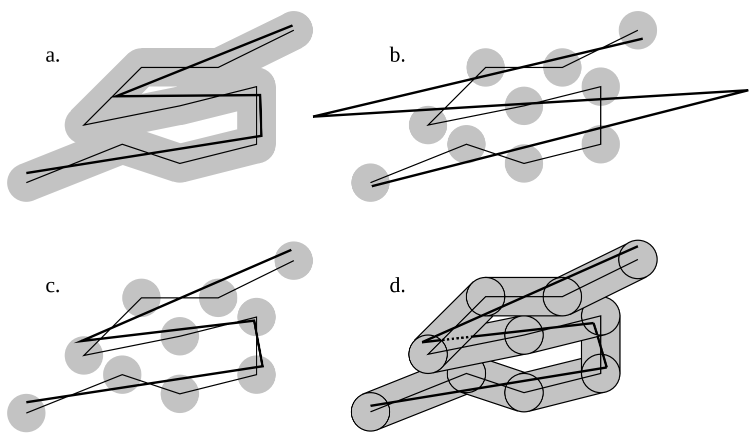

Figure 1: Some approaches to fattening and approximating a polygonal chain

Mathematically, this fattening entails computing the *convolution* of a path, polygon, or subdivision S with a disk (or some other shape) to obtain a region $\mathcal{R}$ in the plane. The convolutions that we are interested in can be computed by several known methods:[16,18] Given the Voronoi diagram[26,42] of the line segments of S, one can compute the convolution $\mathcal{R}$ on a per-cell basis. Alternatively, divide and conquer algorithms can be used.[8,25] Both of these methods run in in $O(n \log n)$ time for convolution by disks or constant size polygons.

In the convolution $\mathcal{R}$, the given polygon or subdivision S defines a homotopy class of curves that can be deformed to S without leaving the region $\mathcal{R}$. We can attempt to find a minimum link representative of the homotopy class. Section 3 makes the definitions for such a "homotopy method" more precise. Its four subsections contain the following results:

Sec. 3.1 We briefly recall the minimum-link path algorithms developed in a previous paper[21] and apply them to approximate paths and polygons. These are greedy algorithms that, after the region $\mathcal{R}$ has been triangulated, find a path in time proportional to the number of triangles that the path passes through.

Sec. 3.2 In contrast, we show that the problem of computing a minimum link subdivision is *NP*-hard. The difficulties comes in optimal placement of vertices of degree three or more; if these are fixed, then we can find the optimum for each chain independently using a minimum link path algorithm.

Sec. 3.3 Returning to polygons, we show that the problem of finding a minimum link simple polygon, that is, one with no self-intersections, is also *NP*-hard.

Sec. 3.4 Given a region $\mathcal{R}$ with h holes, we show that we can find a simple polygon enclosing the holes with at most $O(h)$ links more than the minimum link polygon.

Returning to the line simplification problem, we can see some "features" of this fattening method that are undesirable in some applications. For example, convolution may create quite large regions where the original chain C was dense in the plane; vertices p_i in these regions can be quite far from the approximation $\widetilde{C}$, even though every point of $\widetilde{C}$ is close to C. A simple example is a sharp corner of angle 2θ. If we fatten the segments by ε, the minimum link path can be as far away as $\varepsilon / \sin\theta$—a $10°$ corner can be 11.4ε from the approximation. Also, the convolution itself is difficult to compute robustly.

To address these problems, we consider fattening just the vertices p_i of the chain C by replacing each vertex with a disk of radius ε. We then require that our approximation "visit" each of these disks in order. This method, illustrated in figure 1b, would ensure that vertices of the chain C would be within ε of its minimum link approximation $\widetilde{C}$. If we further restrict the path to turn only inside the vertex disks as shown in figure 1c, then $\widetilde{C}$ would also remain within ε of the original chain C. An alternative shown in figure 1d, which is more in the spirit of the convolution approach and for which minimum link paths are easier to compute, is to convolve each link of C separately with a disk of radius ε, glue the resulting *tubes* at the vertex disks that they share, then compute a minimum link path in this region. Notice that turns are allowed in the tubes and not just the vertex disks, but also that the region formed is not planar—it overlaps itself at every angle.

Section 4 generalizes this approach slightly to a problem we call *ordered stabbing:* given an ordered list of disjoint convex objects, find a polygonal chain that visits the objects in order. We have taken the name from Egyed and Wenger[10], who developed a linear-time greedy algorithm for computing a line stabbing disjoint objects in order, if such a line exists. We extend their algorithm to stabbing with a polygonal chain under three possible restrictions on vertices of the stabber (no restriction, in objects, or in tubes). We also study various definitions of "visiting order" for stabbing disks that may intersect.

Sec. 4.1 We examine Egyed and Wenger's algorithm[10], which uses Graham scan to compute a ordered stabbing line for an ordered set of objects in which consecutive objects are disjoint.

Sec. 4.2 We modify the algorithm to stab intersecting translates of a convex object (e.g. unit disks) with a line, under four definitions of visiting or stabbing order. (The conference version of this paper[17] was incorrect in not restricting the type of intersecting objects.)

Sec. 4.3 We extend the definition of ordered stabbing to polygonal chains. Stabbing line algorithms then give a simple procedure for computing a path that is at most a multiplicative factor of two from the minimum-link ordered stabbing path.

Sec. 4.4 We give a dynamic programming approach to compute the minimum-link ordered stabbing path of intersecting translates of a convex object, when path vertices are not restricted to lie in the translates.

Sec. 4.5 We give a linear-time greedy algorithm to compute the minimum-link ordered stabbing path for a set of objects in which consecutive objects are disjoint.

2. Previous results on approximation

Cartographers have a large catalog of algorithms for the line simplification problem and many measures by which to classify them.[7,29,30] Their algorithms either seek a rough but quick reduction of the data or else an accurate but slow reduction. In comparative tests, the Douglas-Peucker algorithm[9] (also proposed by Ramer[37]) produces the most satisfactory output, but its speed has been criticized.[28,41] The running time of the Douglas-Peucker algorithm has a quadratic worst-case in current implementations, although this can be improved to $O(n \log n)$ worst-case.[20]

A common feature of simplification algorithms is that they use original data points as vertices of the approximation, even these points come from a digitizer with some error. This could be reasonable, except the volumes of data and slowness of accurate reduction algorithms lead to using two or more phases of approximation. In the process of reducing a stream of digitized data to vectors to be plotted on a map, a cartographer may first cast out points until the remaining points are separated by at least ε, and then apply a more complex line simplification algorithm to reduce the data further for storage or display. Though the properties of the individual algorithms are characterized and classified, the properties of these heuristic combinations are not. Criteria much like our ε fattening[6,35,38] are then used *a posteriori* to test the quality of the resulting approximations.

Imai and Iri[22,23,24] and other researchers[2,5,10,19,31,34,40] have chosen mathematical criteria for the approximations and then sought efficient algorithms to find best approximations. The algorithms they have developed, however, have quadratic or greater running times—especially for those that use original data points as vertices of the approximation.

We remove the restriction that vertices of the approximation must be original data points in an attempt to find faster algorithms that fulfill mathematical specifications. Our goal is linear or $O(n \log n)$ algorithms that find the best approximation. Failing that, we may look for a slower algorithm or find a suboptimal approximation—we usually opt for the latter, especially if we can determine how close the approximation is to the optimal.

3. Homotopy classes and minimum link representatives

We begin by studying approximations to polygonal chains and subdivisions that are computed by fattening the original and finding minimum-link paths and subdivisions inside the fattened region.

For this section, we abstract away the method and mechanics of fattening and just suppose that we have a path, polygon, or subdivision S in the plane and a region $\mathcal{R}$ containing S. If we bend and move the components of S, without leaving $\mathcal{R}$, we obtain other paths, polygons, or subdivisions that could be said to be equivalent to S by deformation within $\mathcal{R}$. The topological concept of *homotopy* formally captures this notion of deformation. Let α and β be continuous functions from a topological space S to a topological space $\mathcal{R}$. Functions α and β are *homotopic* if there is a continuous function $\Gamma: S \times [0,1] \rightarrow \mathcal{R}$ such that $\Gamma(s,0) = \alpha(s)$ and $\Gamma(s,1) = \beta(s)$. One can see that homotopy is an equivalence relation.[4,32]

We specialize this definition for paths, polygons, and subdivisions:

- Informally, two paths are *path homotopic* if one can be deformed to the other in $\mathcal{R}$ while keeping the endpoints fixed. Formally, we set $S = [0,1]$ and find a function Γ where $\Gamma(0,t)$ and $\Gamma(1,t)$ are the two paths and $\Gamma(s,0)$ and $\Gamma(s,1)$ are the endpoints of the paths.

- A polygon is the image of a circle S^1 under a continuous map into $\mathcal{R}$. Two polygons with maps α and β are *homotopic* if there is a continuous map $\Gamma: S^1 \times [0,1] \rightarrow \mathcal{R}$ such that $\Gamma(x,0) = \alpha(x)$ and $\Gamma(x,1) = \beta(x)$.

- Two subdivisions α and β in $\mathcal{R}$ are homotopic in $\mathcal{R}$ if α can be deformed to β within $\mathcal{R}$.

If the fattened region $\mathcal{R}$ is actually obtained by convolving the path, polygon, or subdivision S with a disk of radius ε (that is, by drawing it with a fat brush) then the minimum-link object homotopic to S not only remains within ε of the original, but can also be deformed to the original while remaining within ε.

The homotopy class can be represented to a computer by giving a representative path, such as the Euclidean shortest path that is homotopic to S. A more useful representation for computation comes from triangulating the the region $\mathcal{R}$ so that all of the vertices of the triangulation are on the boundary. The homotopy class of S can be represented by the sequences of triangles and triangulation edges that the curves of S intersect.

3.1. Computing minimum link paths and polygons of a given homotopy type

Hershberger and Snoeyink[21] have investigated computing minimum link paths and closed curves of a given homotopy class in triangulated polygons. (They also consider minimum length and restricted orientations.) They prove:

Theorem 1 *One can compute a minimum link path α' that is homotopic to a chain α in time proportional to the number of links of α and the number of triangles intersecting α and α'. In the same time, one can compute a closed polygon α' homotopic to a polygon α and having the minimum number of links if α' is non-convex or at most one more than the minimum number if α' is convex.*

These paths and polygons are computed by a greedy procedure, following Suri[39] and Ghosh.[13] In brief, the idea is to illuminate as much of the region $\mathcal{R}$ as possible from the starting endpoint of a path; this is as far as one link can reach. Repeat the illumination from the appropriate boundary of the lit area until the goal point is found. The "appropriate boundary" is determined by the homotopy class of α—or more specifically, by the triangulation edges crossed by the Euclidean shortest path of the homotopy class of α. Thus, we have an algorithm whose running time is a small polynomial in the complexity of the fattened region and the input and output paths.

3.2. The min-link subdivision problem is NP-hard

Given a subdivision S in a polygonal region, P, the min-link subdivision problem (MinLinkSub) asks for the polygonal subdivision S' homeomorphic to S in P that is composed of the minimum number of line segments. This problem is related to simplifying an entire map. We look at the decision problem to show that MinLinkSub is *NP*-hard: Given S and P and an integer k, is there a polygonal subdivision S' with at most k segments that is homeomorphic to S in P?

First, we note that the planar case of a problem that Garey, Johnson and Stockmeyer[12] have called maximum 2-sat (Max2Sat) is *NP*-complete. The general case of Max2Sat is: Given a set of variables V, an integer k, and disjunctive clauses $C_1, C_2, \ldots, C_p$, each containing one or two variables, determine if some truth assignment to the variables satisfies at least k clauses. The *variable graph* of an instance of Max2Sat is defined to be the graph $G = (V, E)$, with an edge $(u, v) \in E$ if and only if the variables u and v appear together in some clause C_i. An instance of Max2Sat is *planar* if its variable graph is planar.

Theorem 2 *Planar maximum 2-sat (Max2Sat) is NP-complete.*

Proof: One can guess a truth assignment and, in linear time, verify that at least k clauses are satisfied. Thus, planar Max2Sat is in *NP*.

Garey, Johnson, and Stockmeyer[12] prove that Max2Sat is *NP*-hard by reducing 3-sat to Max2Sat. Their reduction preserves planarity, so we use it to reduce planar 3-sat to planar Max2Sat and show that the latter is also *NP*-hard.

Consider an instance of planar 3-sat with m clauses. Since we can duplicate variables, we can assume that each clause has three variables. Construct an instance of Max2Sat by replacing every clause $(a_i \vee b_i \vee c_i)$ with ten clauses (a_i), (b_i), (c_i), (d_i), $(\overline{a_i} \vee \overline{b_i})$, $(\overline{b_i} \vee \overline{c_i})$, $(\overline{a_i} \vee \overline{c_i})$, $(a_i \vee \overline{d_i})$, $(b_i \vee \overline{d_i})$, and $(c_i \vee \overline{d_i})$. At most six of these clauses can be satisfied if the original was not—seven can be satisfied if the original was. Thus, a total of $7m$ 2-sat clauses can be satisfied if and only if all m 3-sat clauses can be satisfied.

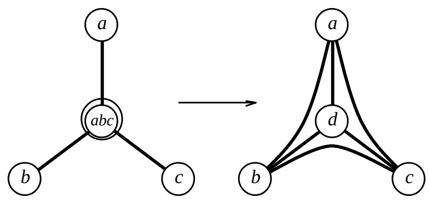

Figure 2: From planar 3-sat to planar Max2Sat

Given a planar embedding of the clauses and variables of the 3-sat instance, we form a planar embedding of the Max2Sat variable graph by replacing the clause $(a_i \vee b_i \vee c_i)$ with the variable d_i as shown in figure 2. Since planar 3-sat is *NP*-hard [11,27], planar Max2Sat is, too. □

We prove that the minimum link subdivision problem, MinLinkSub, is *NP*-hard by taking an instance of planar Max2Satand constructing an instance of MinLinkSub that has a solution if and only if the instance of Max2Sat has a solution. Let us take an informal look at the gadgets for truth assignments and for unary and binary clauses that are used in the construction. We embed the variable graph of the 2-sat instance in the plane with straight-line edges such that no edge is vertical. Then we fatten each vertex to a disk and each edge to a rectangular strip build our MinLinkSub instance within the resulting region. Within each disk we place *true* and *false* points, directly above and below the disk center, and force the vertex of the minimum-link subdivision to lie at one of these points by using appropriate gadgets.

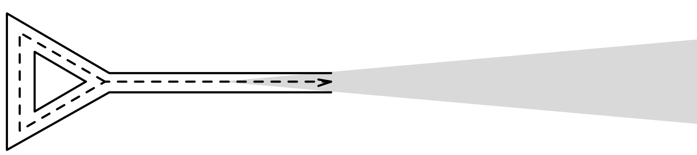

Figure 3: An enforcer and its cone

For a unary clause, we add an *enforcer* pointing to the *true* point for a positive clause and the *false* point for a negative clause. Figure 3 illustrates an enforcer—dashed lines are subdivision edges and solid lines are region boundaries. The enforcer can be realized by four line segments if and only if the subdivision vertex lies in the shaded cone.

For the binary clauses on two variables, we divide the rectangular strip of the fattened edge joining the two variables into four strips. In each we form *negaters* for variables that need them and a *gate* to simulate an OR gate. Figure 4 illustrates a negater and gate combination for the clause $(\overline{a} \vee b)$—the dashed line is the subdivision edge, solid lines are region boundaries, and grey lines are possible satisfying assignments.

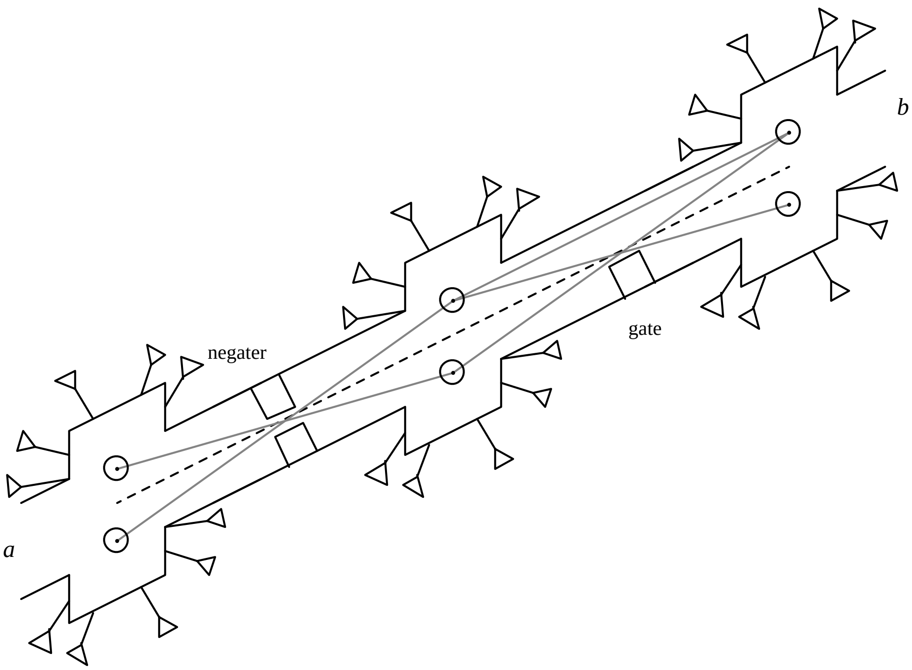

Figure 4: A negater and gate combination for $(\overline{a} \vee b)$

In a minimum link subdivision, each clause that is not satisfied requires one extra line segment. Thus, there is a number, k', such that k clauses of the instance of Max2Sat can be satisfied if and only if the instance of MinLinkSub uses at most k' line segments. Theorem 3 shows that this construction can be carried out.

Theorem 3 *MinLinkSub is NP-hard.*

Proof: We prove that MinLinkSub is *NP*-hard by a reduction from Max2Sat.

Embed the variable graph, G, of a Max2Sat instance in the plane with straight edges such that no edge is vertical. Let θ_{min} be the minimum angle between edges, θ_{vert} be the minimum angle of an edge from vertical, d_{e} be the length of the shortest edge, and d_{ve} be the shortest distance from a vertex to a non-incident edge. Table 1 lists these and other important dimensions of the construction, figure 5 illustrates them, and table 2 gives the relations between them.

We fatten the vertices of G to *vertex disks* of radius r_{v} and edges to strips of width w. This fattening *preserves the face structure of* G if there is a one-to-one correspondence between the faces of G and the connected components of the complement of its fattening such that a face bounded by a sequence of edges and vertices maps to a component bounded by portions of the disks from the same edges and vertices in the same sequence. Conditions 2–4 in table 2 ensure

Variable	Description
d_{e}	Length of shortest edge
d_{ve}	Shortest distance from a vertex to a non-incident edge
$\theta_{\min}$	Minimum angle between two edges
θ_{vert}	Minimum angle of an edge from vertical
r_{v}	Radius of *vertex disks* (fattened vertices)
w	Width of fattened edge
r_{b}	Radius of *boolean disks* (e.g. the ball around a *true* point)
h	Height of *true* point above a vertex
θ_{enf}	Minimum angle between two enforcers
c	Enforcer cone width at r_{v}

Table 1: Variables for the construction, illustrated in figure 5

#	Constraint	Reason
1.	d_{e}, d_{ve}, $\theta_{\min}$, θ_{vert}	Given by the embedding
2.	$0 < r_{\mathrm{v}} < d_{\mathrm{e}}/8$	Vertex disks don't engulf edges
3.	$0 < w < r_{\mathrm{v}} \tan(\theta_{\min}/2)/2$	Vertex disks appear on face between adj. edges
4.	$r_{\mathrm{v}} + w < d_{\mathrm{ve}}$	No edge & vertex become incident by fattening
5.	$\tan(\theta_{\mathrm{enf}}/2) > c/r_{\mathrm{b}}$	No point outside a bool. disk is in three cones
6.	$2(h + r_{\mathrm{b}}) < w/8$	Boolean disks are visible along fattened edges
7.	$\sin(\theta_{\mathrm{vert}}/2) > r_{\mathrm{b}}/h$	Slopes that intersect both bool. disks $< \theta_{\mathrm{vert}}/2$

Table 2: Constraints on the variables. (See table 1 and figure 5.)

that the fattening of G by r_{v} and w preserves the face structure.

Within each vertex disk we place *true* and *false* points, h above and h below the vertex. Around each point, we draw a *boolean disk* of radius r_{b}. We can force the vertex of a minimum link subdivision to lie in one of these two boolean disks by using *enforcers,* each consisting of a path from the vertex to a small triangle such that the path can be a single line segment if the vertex lies inside the *enforcer cone* as illustrated in figure 3. The enforcers are placed around the vertex disks; the cones can be made to have radius at most c at distance r_{v} by moving the walls of the enforcer together.

For a variable used in k binary clauses, we add $k+3$ enforcers pointing to each boolean disk. If condition 5 holds, then cones from enforcers pointing to the same boolean disk do not intersect outside the disk; this implies that any point outside the boolean disks lies in at most two enforcer cones. In a minimum link subdivision, each subdivision vertex is placed in a boolean disk because placement at any other point causes at least $2k+4$ enforcers to have an extra line segment, while placement at a *true* or *false* point adds $k+3$ segments to enforcers and at most k to clauses. Thus, the placement of a vertex in a minimum link subdivision can be interpreted as a truth assignment.

Next we form clauses. For a unary clause, we simply add another enforcer pointing to the *true* point for a positive clause or the *false* point for a negative

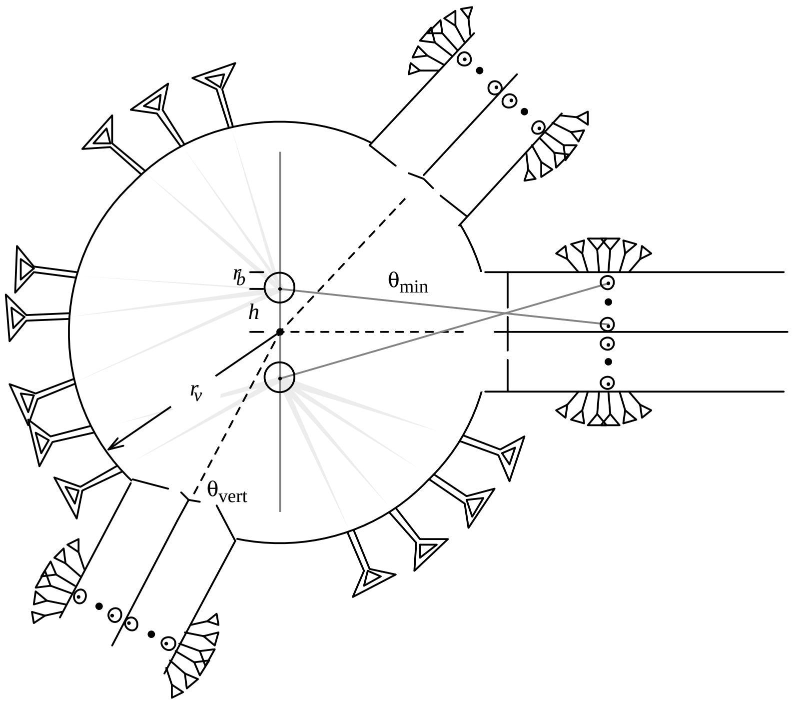

Figure 5: Variables in the construction, described in table 1.

clause.

For the binary clauses on a given pair of variables, we divide the fattened edge into four strips and form boolean balls at both ends of each strip. Then, for a clause (an OR gate) with both variables positive, we add a block within a strip so that the edge can be a single line segment if and only if one of the incident vertices is placed in a *true* disk. When both variables are negative, we add the block so that one of the vertices must be placed in a *false* disk. When the variables differ in sign, we pair an OR gate with a *negater* for the negative literal as shown in figure 4. Condition 6 ensures that satisfiable edges can be represented by one segment and condition 7 ensures that unsatisfiable edges require two segments.

Each clause that is not satisfied adds one extra line segment to the minimum link subdivision. Thus, there is some k' such that k clauses of the instance of Max2Sat can be satisfied if and only if the instance of MinLinkSub uses at most k' line segments. □

Placement of vertices of degree at least three is the difficult part of MinLinkSub. If one could guess the locations of the vertices of degree three or greater, then one could use a minimum-link path algorithm[39,13,21] to find paths joining adjacent vertices: Connect each guessed vertex to its original by a path using at most n links. Then, for every pair of adjacent vertices, a and b, compute the minimum link path homotopic to the path that goes from guessed a to original a to original b to guessed b. The path algorithm performs this computation in polynomial time in the size of its input.

3.3. Minimum-link simple polygons

A desirable and natural restriction to add to the line simplification problem is that boundaries that do not self-intersect should not be made to self-intersect. That is, simple polygonal chains should be replaced by simple polygonal chains. In constrast to the polynomial time algorithms of theorem 1, we show that the problem of finding a minimum-link simple polygon of a given homotopy type (MinLinkSP) is NP-hard by a reduction from planar Max2Sat.

The reduction is much like the one used in section 3.2: We embed an Euler tour of the variable graph as a simple closed curve in the plane and place obstacles so that graph vertices are pinned in place. Then we form toggle switches at each graph vertex and use enforcers to ensure that an approximate path can be interpreted as a truth assignment. Finally, we arrange negaters and gates so that an edge of the graph can be embedded using fewer links if the clause is satisfied.

Theorem 4 *MinLinkSP is NP-hard.*

Proof: Suppose we are given an of Max2Sat. We puncture the plane by a polynomial number of holes and construct a simple (non-self-intersecting), representative curve of a homotopy class of curves in the resulting region. We prove that there is a simple, polygonal curve in this homotopy class having fewer than k' line segments if and only if the instance of Max2Sat has a truth assignment satisfying at least k clauses.

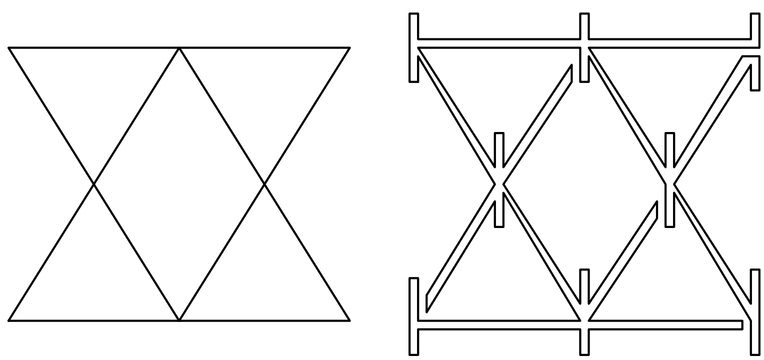

Figure 6: A graph and its edge polygon

Embed the variable graph of the Max2Sat instance in the plane so that no edge is vertical. Add short vertical edges just above and below each vertex. By splitting vertices, we can form a planar tree that contains all the edges of the variable graph; an *edge tree*. A walk around the edge tree gives us the *edge polygon*, a simple polygon in which each clause edge appears twice. See figure 6.

In subsection 3.2, we constructed a region in which the number of edges of a minimum link embedding of a variable graph was a function of the number of

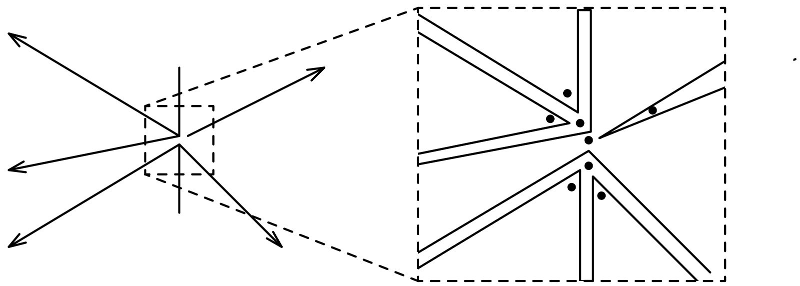

Figure 7: Vertex gadgets

satisfiable clauses. Here we embed the edge polygon in a manner that mimics the variable graph embedding. For a vertex of degree d, we add a *vertex gadget* of $\leq 2d$ holes that the edge polygon must wind through as shown in figure 7.

We also add pentagonal holes above and below each vertex and turn the short vertical paths that we added to the edge polygon into two *toggles*, which together form a *switch*. The homotopy class of the lower toggle is illustrated in figure 8. Qualitatively, the path from the vertex zig-zags among some holes on on the left that we call *enforcers*, then goes back and forth across the pentagon, with more holes to hold it in place, and then among more enforcers on the right. The upper toggle lacks enforcers, but otherwise is symmetric through the vertex.

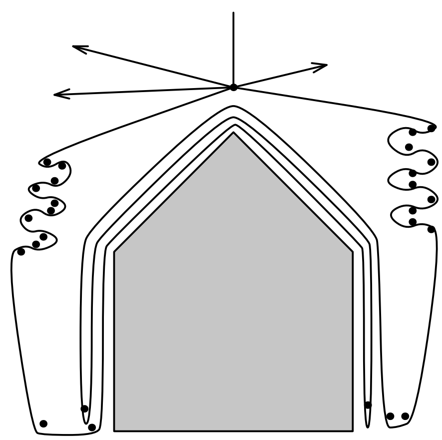

Figure 8: The homotopy class of a toggle

Each path across a pentagon can consist of as few as three line segments. In a minimum link embedding with no self intersections the paths across both pentagons must be nearly parallel, as shown in figure 9, to be realized with three segments. We then say that the toggles have parallel slants. The enforcers on the lower toggle encourage both toggles to slant to the extreme right or left. A switch with toggles slanting down to the right is considered set *true;* slanting left is considered *false.*

The switch for a vertex corresponding to a variable that appears in k unary and binary clauses has $2k+1$ enforcers on each side of the lower toggle so that any slant other than extreme right or extreme left adds extra segments to $4k+2$ enforcers, whereas an extreme slant adds segments to $2k+1$ enforcers and at most $2k$ to edges. Each toggle path goes back and forth at least $6k+3$ times so that adding segments to enforcers and edges is preferable to adding segments to a toggle. Thus, in a minimum link path with no self-intersections, each switch is unambiguously *true* or *false*. Because two holes are sufficient for each toggle path and four for each enforcer, the number of holes required is less than $40(k+1)$.

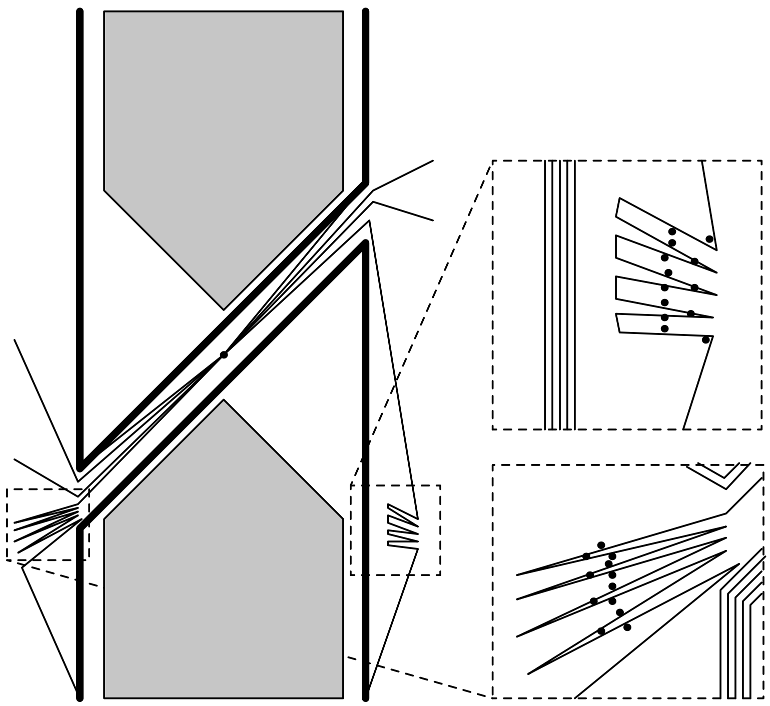

Figure 9: A switch with enforcers

To simulate a unary clause, we add two extra enforcers to the true (or false) side of the lower toggle so that each enforcer each require an additional segment if the clause is not satisfied. Binary clause OR gates are simulated by blocking the appropriate segment, just as in subsection 3.2; negaters use two blocks. Figure 10 illustrates a single binary clause—the grey lines are possible embeddings of satisfying assignments and the dashed is the embedding of an unsatisfying assignment. Since each edge of the variable graph is doubled in forming the edge polygon, any binary clause that is not satisfied by a truth assignment requires two extra line segments in the minimum link simple polygon.

Thus, there is a number, k', such that k clauses of the instance of Max2Sat can be satisfied if and only if the instance of MinLinkSP, the minimum link simple polygon problem, uses at most k' line segments. $\square$

One can break the polygon inside one of the vertex gadgets and anchor its endpoints to obtain a path. Thus, the minimum link simple path problem is also *NP*-complete.

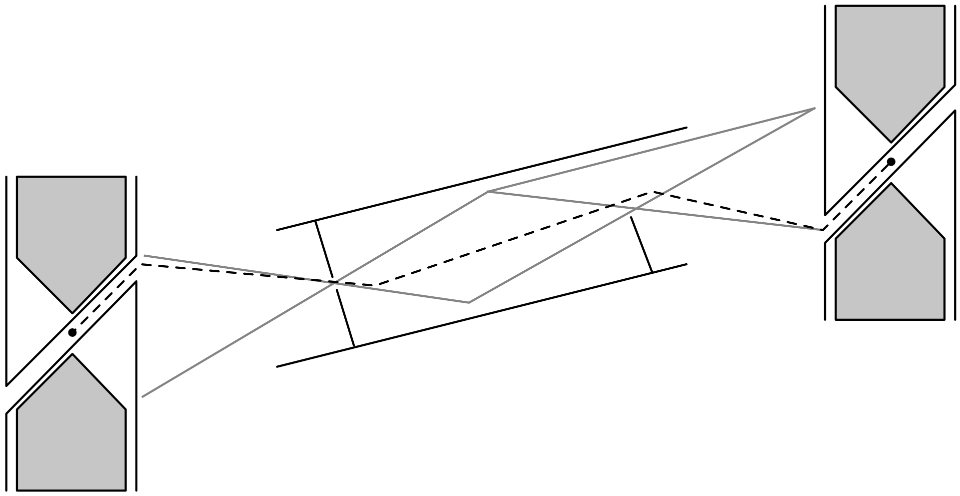

Figure 10: A negater and gate combination

3.4. Minimum link simple curves enclosing all holes

The reduction in the previous section requires holes both inside and outside the curve; whether one can efficiently find a minimum link simple curve in a polygon with h holes that encloses all the holes is an open question. We can find a simple curve that has only $O(h)$ more segments than the (non-simple) minimum link curve; this is independent of the number of segments of the minimum link curve. We identify $O(h)$ *junction triangles* of the triangulation and group the rest of the triangles into *corridors*. In each corridor we find the minimum link path.

Theorem 5 *In a polygon P with n vertices and h holes, one can, in $O(n)$ time, find a simple closed curve enclosing all the holes that has $O(h)$ segments more than the minimum link curve of the same homotopy class.*

Proof: Let α' be the Euclidean shortest curve homotopic to α—the relative convex hull of the holes. The curve α' intersects any triangulation edge at most twice.

Because all the holes are inside of α', the curve α' does not cross any triangulation edge between two holes. We cut along any edges between two holes, forming bigger holes. Because the original holes do not intersect, the number of cuts around the boundaries of the new holes is bounded by the length of a circular Davenport-Schinzel sequence with at most three alternations.[1] Thus, there are at most $2h - 2$ cuts.

Call any triangle in which α' crosses all three sides a junction triangle. There are two types: *three-way junctions*, in which all vertices lie on the outer boundary and *two-way junctions*, in which two vertices lie on the outer boundary and one lies on a hole.

Removal of a three-way junction triangle leaves three connected components, each of which must have a hole. One can form a three-way tree whose leaves

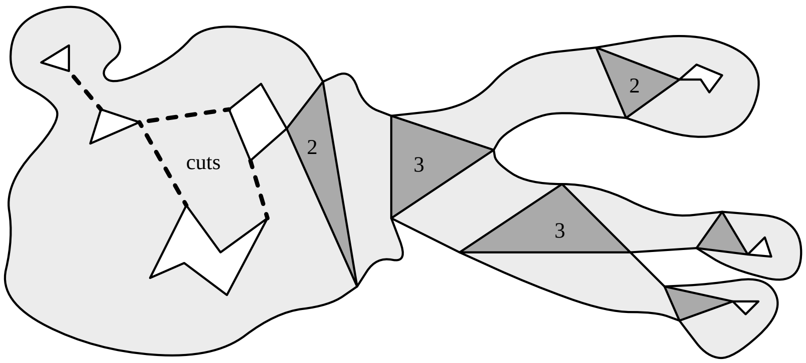

Figure 11: Cuts and junction triangles bound corridors

contain holes and whose internal nodes are three-way junctions such that the holes of a component formed by removing a junction are all in the same subtree. This implies that the number of three-way junctions is at most $h-2$. Furthermore, one can cut the edge of a two-way junction that goes from outer boundary to outer boundary to separate P into two components, each of which has a hole and in one of which the hole has a vertex of the junction triangle. A particular partition of holes can happen in only two ways, so there are only $2h$ two-way junction triangles.

The triangles with at least one vertex on the outer boundary can now be grouped into maximally connected corridors, bounded by junction triangles and cuts, through which the shortest path α' passes one or two times. Within each corridor, C, we find the minimum link path β_C that goes from p_C, the midpoint of one bounding junction triangle, to q_C, the midpoint of the other, using a minimum link path algorithm as discussed in section 3.1.

The minimum link path β_C may require more segments than the minimum link path from p_C to q_C of the same homotopy type because the latter path may cross cuts that bound the corridor. A path that crosses a cut, however, does so an even number of times. By connecting the first and last crossing with a portion of the cut, we obtain a path that remains within the corridor and has only as many additional segments as there are cuts bounding the corridor. As we argued above, the number of cuts bounding all corridors is at most $2h-2$.

Finally, we link up the paths through corridors into a closed curve β in the homotopy class of α. The curve β gains at most two segments more than the minimum curve through corridors for each junction triangle that it passes through. Thus, β is within $O(h)$ line segments of the minimum link closed curve enclosing the h holes. $\square$

The worst case for our procedure has no cuts, $h-2$ three-way junctions and h two-way junctions. This results in $10h-12$ additional line segments. We have yet to find a polygon that requires more than $2h-2$ additional segments to make a minimum link curve simple.

4. Ordered Stabbing

In this section, we study the *ordered stabbing* problem: Given an ordered sequence of n convex objects, $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$, find a polygonal chain, consisting of the minimum number of line segments, that *visits* the objects in order. Different variants of the ordered stabbing problem arise from restrictions on the stabbed objects or stabbing path as well as from different definitions of "visiting order." We consider several variants in the following subsections, emphasizing those for which there are efficient greedy algorithms.

4.1. Ordered stabbing of disjoint objects with a line

Egyed and Wenger[10] looked at the problem of stabbing disjoint convex objects in order with a line. They show that the actual shape of the objects matters less than the ability to find inner and outer common tangents—if one assumed that computing these tangents took constant time, then one could find a line stabbing the objects in order by a simple Graham scan. We reinvent (and simplify) their algorithm for stabbing disjoint objects with a line and extend it in later subsections.

It may help to think about a simple instance of ordered stabbing: Is there a line stabbing a set of vertical segments ordered by x-coordinates? To answer this question, one can form the convex hulls of the "above" endpoints of segments and the hull of the "below" endpoints. If these hulls are separable—if they have inner common tangents, for example—then and only then does a stabbing line exist. We define support hulls and limiting lines to allow us to use this method for stabbing more general objects.

If α is a direction, then let $-\alpha$ denote the reverse direction. We call an object $O \in \mathcal{O}$ a *support object* for direction α if there is a line ℓ_α in direction α such that O lies on and to the left of ℓ_α and no object $O' \in \mathcal{O}$ lies strictly to the left of ℓ_α. The support object in figure 12 is shaded. The line ℓ_α is called a *support line* for direction α and the point or points of $O \cap \ell_\alpha$ are called *support points*. We can observe the following connection between support lines and stabbing lines.

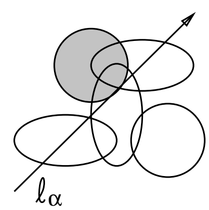

Figure 12: Support line ℓ_α

Observation 1 *The lines parallel to direction α that stab a set of objects $\mathcal{O}$ are exactly the lines to the left of both support lines ℓ_α and $\ell_{-\alpha}$, if any exist.*

By analogy with the convex hull of segment endpoints, we can define the *support hull* of a set of n objects as the circular list of support objects, ordered by the angles of their support lines. Repetitions are possible, as figure 13 shows, but if any two objects O and O' have at most two outer common tangents, then any subsequence of the list can have only two alternations between O and O'. Thus, the size of the list is at most $2n-2$ by Davenport-Schinzel sequence bounds[1].

A support line ℓ_α is a *limiting line* if its reverse $\ell_{-\alpha}$ is also a support line, as shown in figure 13. Limiting lines are analogous to inner common tangents. A limiting line ℓ_α hits two support points; we name them the *first contact*, p, and *second contact*, q, so that the vector $q-p$ has direction α. We name the objects that contain these points the *first* and *second contact objects* for ℓ_α, respectively. We can distinguish two types of limiting lines: ℓ_α is a *counterclockwise (ccw) limiting line* if the first contact p is the support point for ℓ_α, as shown in figure 13, and a *clockwise (cw) limiting line* if the second contact q is the support point for ℓ_α.

Limiting lines are stabbers, as in figure 13, but rotating a ccw limiting line counterclockwise gives a line that is no longer a stabber. In our ordered stabbing problems, we will find at most one limiting line of each type; they will delimit the possible slopes for stabbing lines. The above and below portions of the support hull between these slopes limit the extent that a stabbing line can move up and down. Thus, the hulls and limiting lines give a linear size description of all possible stabbers. In the rest of this section, we show how to maintain this description under the assumption that basic operations, such as computing the intersection of an object with a line and computing common tangents of two objects, take constant time. We prove the following theorem.

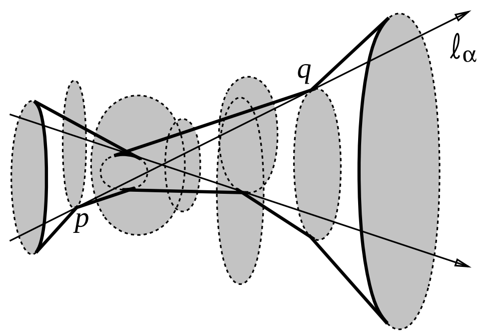

Figure 13: Support hull with limiting lines

Theorem 6 *Let $\mathcal{O} = \{O_1, O_2, O_3, \ldots\}$ be a sequence of convex objects in which consecutive objects are disjoint. One can compute a line that stabs the longest possible prefix $O_1, O_2, \ldots, O_i$ in order using $O(i)$ time and space.*

Proof: We outline the idea; algorithm 1 gives more complete pseudocode.

Assume that a vertical line separates the first two objects with O_1 left of O_2 as in figure 14. We can easily compute a description of all ordered stabbers for O_1 and O_2: Initialize the ccw limiting line t and the cw limiting line t' to the appropriate inner common tangents directed from O_1 toward O_2. Two portions of the support hull have slopes that fall between the slopes of t and t'; these portions are delimited by the contact points of t and t'. We name them the *above hull*, A, and the *below hull*, B, as shown. To represent A and B, we store the list of support objects in a *deque*—a doubly-ended queue—which we will maintain by a Graham scan procedure.[15] Initially, both deques contain O_1 at the tail and O_2 at the head.

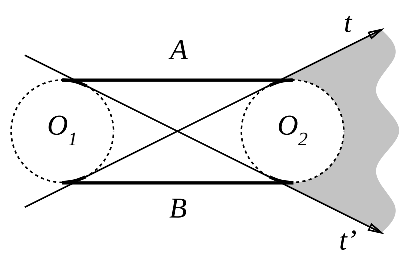

Figure 14: Initial description

We would like to add objects successively and maintain the description of ordered stabbers. Given above and below hulls A and B for the first i objects and

Data Structures: Store the above support hull A, in a deque that supports the following in constant time: The operations **Push**(A, end, O_i) and **Pop**(A, end) push and pop objects from the head or tail of A, depending on whether *end* is *head* or *tail*. Pointers **Tail**(A), **NTail**(A), **NHead**(A) and **Head**(A) are maintained to the tail (lowest index) next-to-tail, next-to-head, and head (highest index) objects in A. Store B similarly.

Initialization: Place object O_1 at the tail and O_2 at the head of both A and B and set limiting lines t and t' to the ccw and cw inner common tangents. Then set $i := 2$ and execute the following algorithm to add O_{i+1}.

1. While O_{i+1} intersects the wedge between t and t' and right of O_i do
- **2.** If O_{i+1} does not intersect t then
  - (* *Update the head of support hull* A *)
  - **3.** While **Head**(A) is above the higher outer common tangent from **NHead**(A) to O_{i+1} do
    - **4.** **Pop**($A, head$)
  - **5.** EndWhile
  - **6.** **Push**($A, head, O_{i+1}$)
  - (* *Update ccw limit line* t *and the tail of support hull* B *)
  - **7.** Set t to the ccw inner tangent from **Tail**(B) to O_{i+1}
  - **8.** While **NTail**(B) is not below t do
    - **9.** **Pop**($B, tail$)
    - **10.** Set t to the ccw tangent from **Tail**(B) to O_{i+1}
  - **11.** EndWhile
- **12.** EndIf
- **13.** If O_{i+1} does not intersect t' then
  - (* *Update the head of support hull* B *)
  - **14.** While **Head**(B) is below the lower outer common tangent from **NHead**(B) to O_{i+1} do
    - **15.** **Pop**($B, head$)
  - **16.** EndWhile
  - **17.** **Push**($B, head, O_{i+1}$)
  - (* *Update cw limit line* t' *and the tail of support hull* A *)
  - **18.** Set t' to the cw inner tangent from **Tail**(A) to O_{i+1}
  - **19.** While **NTail**(A) is not above t do
    - **20.** **Pop**($A, tail$)
    - **21.** Set t' to the cw tangent from **Tail**(A) to O_{i+1}
  - **22.** EndWhile
- **23.** EndIf
- **24.** Set $i := i + 1$.

25. EndWhile

Algorithm 1: The basic algorithm for the ordered stabbing of disjoint objects with a line limiting lines t and t', we want to add object O_{i+1}. We define the *line-stabbing wedge* to be the region between t and t' that is right of object O_i—drawn shaded in figures 14 and 15. For every point p in the line-stabbing wedge there is a line through p that visits the first i objects before visiting p. If O_{i+1} does not intersect the wedge, then no stabbing line visits the first $i+1$ objects in order. If it does, then we update the limiting lines, which are ordered stabbing lines, and the portions of the support hull.

If the ccw limiting line t does not intersect object O_{i+1}, then we must move t clockwise until it does. We also update the head of the above hull list A by Graham scan. Specifically, to add object O_{i+1} to A, some suffix may first need to be removed as in lines 3 to 6 of algorithm 1. Furthermore, the first contact object of t in B may change during the motion. If it does, the old contact is removed from the tail of B by line 9. The cw limiting line t' is handled similarly.

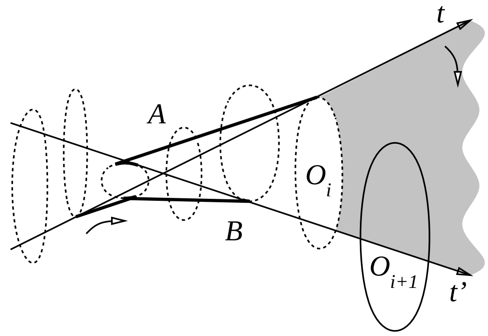

Figure 15: Updating the wedge and hulls

All operations performed when O_{i+1} is added take constant time except for deque maintainence. Since an object is added to each deque once and removed at most once, the total computation is linear in the number of objects considered. □

Remark: We described the algorithm as started at the beginning of the sequence of objects and always adding objects to the end. Because adding objects to the tail (in reverse sequence, of course) is symmetrical, one could begin in the middle and add to both sides.

4.2. Ordered stabbing of intersecting unit disks with a line

In this section, we extend algorithm 1 to stab an ordered set of possibly intersecting unit disks with a line. Our algorithm can be applied to translates of a constant-sized convex polygon as well—unit squares, for example, which arise when ε-disks are computed in the L_1 or L_∞ metrics. We continue to say "disks" for convenience.

We consider four possible definitions of *visiting order* for intersecting objects. All four definitions are equivalent to the natural definition if the objects are disjoint. Given two points p and q on a directed line ℓ, we say that $p \prec q$ if the vector from p to q is in the direction of ℓ. Let the intersection $\ell \cap O_i$ have extreme points $a_i \prec b_i$. Given a sequence of objects $O_1, O_2, \ldots, O_n$ and a line ℓ such that the intersection $\ell \cap O_i$ has extreme points $a_i \prec b_i$, we say that ℓ *visits the objects in order* if

Def. 1: Line ℓ exits the objects in the correct order: For $i < j$, we have $b_i \prec b_j$.

Def. 2: Line ℓ enters the objects in the correct order: For $i < j$, we have $a_i \prec a_j$.

Def. 3: Line ℓ both enters and exits the objects in the correct order: For $i < j$, we have $a_i \prec a_j$ and $b_i \prec b_j$.

Def. 4: Line ℓ hits points $p_1, p_2, \ldots, p_n$, with $p_i \in \ell \cap O_i$, in the correct order: For $i < j$, the point $p_i \prec p_j$.

Definitions 1 and 2 could be considered equivalent: given an algorithm that computes stabbing lines for one definition we can compute stabbing lines for the other by reversing the sequence of objects. We will, however, combine the algorithms for 1 and 2 to handle definition 3. Since the algorithms that compute stabbers without reversing the sequence are slightly different, we treat definitions 1 and 2 separately. Definition 4 is perhaps the most natural and, as we will see in section 4.3, is related to the Fréchet metric.

As in the previous section, our task is to maintain support hulls and line-stabbing wedges as we consider disks $O_1, O_2, \ldots, O_n$ in sequence. The wedge for $O_1, O_2, \ldots, O_i$, which is the locus of all points p such that some line visits $O_1, O_2, \ldots, O_i$ in order and then visits p, depends on the definition of visiting order.

Theorem 7 *Let $\mathcal{O} = \{O_1, O_2, O_3, \ldots\}$ be a sequence of unit disks or translates of a constant-size convex polygon. One can compute a line that stabs the longest possible prefix $O_1, O_2, \ldots, O_i$ using $O(i)$ space and $O(i)$ time for visiting order definitions 1–3 or $O(i \log i)$ time for definition 4.*

Proof for Def. 1: Let us begin with definition 1: exiting the disks in the correct order. A way to view the result that we are trying to obtain is to imagine that the disks are painted on the plane in reverse order—starting with disk O_n. An ordered stabbing line must exit a visible portion of the boundary of each disk. We will not compute this "painting;" it will, however, guide us in modifying algorithm 1 to add disk O_{i+1} and update the description of the stabbers of the first i disks. First we outline how to maintain this description, then how to initialize it.

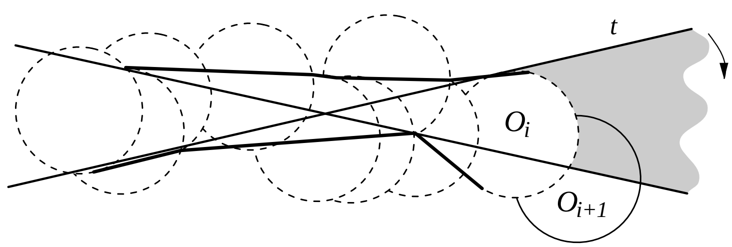

Figure 16: Updating the wedge under definition 1

To add O_{i+1}, we must determine the ordered stabbers of $O_1, \ldots, O_i$ that exit O_{i+1} after O_i. As before, the line-stabbing wedge is the region between the limiting lines t and t' and right of O_i. Because O_i is exited last, no disk O_j with $j < i$ intersects the wedge. Also as before, if O_{i+1} does not intersect the wedge then no stabbing line exists.

In our imaginary painting, O_{i+1} may be obscured by O_i; thus, we discard portions of O_{i+1} that lie outside the line-stabbing wedge. By restricting our objects to translates of a given object, we can be assured that what remains of O_{i+1} is connected. If what remains does not intersect the ccw limiting line t, then we must update the support hulls and the line t.

First update the head of the above hull A, as in lines 3 to 6 of algorithm 1. If O_i and O_{i+1} intersect, then their upper intersection point may become a point on the support hull, as will occur in figure 16. To this end, the tangent from this intersection point to **NHead**(A) must be considered in line 3 and the deque data structure must be extended to store support points as well as support disks.

Once the support hull A is updated, t moves clockwise until it comes to rest on the disk or point that is last in A. This may cause disks to be removed from the tail of B as in line 9. The cw limiting line is adjusted in a similar fashion.

What remains is to initialize the description of ordered stabbers. We can reuse the description of figure 14 if O_1 and O_2 do not intersect. If they do intersect, the description is rather strange. The above hull A consists of O_1 follwed by the upper intersection point of $O_1 \cap O_2$; the below hull B of O_1 and the lower intersection point. The limiting lines, then, are tangents from an intersection point to O_1 that cannot be rotated further as shown in figure 17. The wedge they form is greater than 180° so angle comparisons must be performed carefully. This adds to the programming complexity, but not the asymptotic time complexity. □

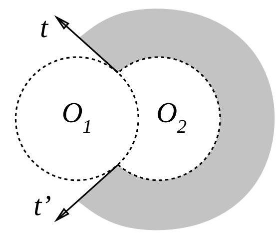

Figure 17: Initial wedge, definition 1

Proof for Def. 2: Stabbing lines satisfying definition 2, entering the disks in the correct order, must hit the boundaries of disks in a "painting" that starts with O_1. They can be found by a similar algorithm.

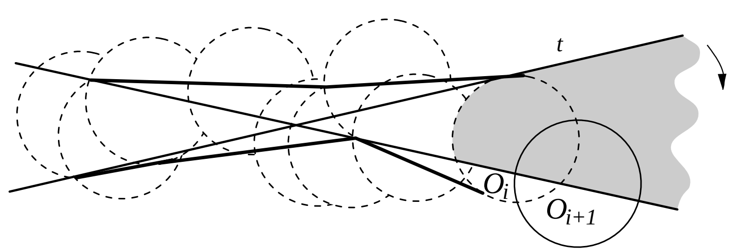

Figure 18: Updating the wedge under definition 2

Define the line-stabbing wedge to be the convex region bounded by the two limiting lines and not left of O_i. In figures 18 and 19, the line-stabbing wedges are shaded. Any stabber that crosses into the wedge has already entered every disk up through O_i. Thus, we need to determine and discard the stabbers that enter O_{i+1} and O_i in the wrong order.

Following the painting model, discard portions of O_i that lie inside O_{i+1}. If the remaining portion of O_i no longer intersects the ccw (or cw) limiting line, or if O_{i+1} does not intersect the line, then we must update the support hull and limiting line as before. We again use a Graham scan to maintain support points and support disks in A and B with the key property that the support points or disks for the limiting lines are the first and last entries in A and B.

The initial hulls and limiting lines of figure 14 can be reused if O_1 and O_2 do not intersect. If they do intersect, the initial support hulls A and B consist of the upper and lower intersection points, respectively, followed by O_2. The limiting lines are tangents to O_2 from the intersection points, as shown in figure 19. Again, the wedge is greater than 180°. □

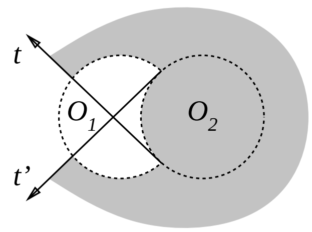

Figure 19: Initial wedge, definition 2

Proof for Def. 3: We can combine the two previous algorithms to find stabbing lines satisfying definition 3. Given the support hulls A and B and limiting lines after the first i disks we need to determine the ordered stabbing lines that enter and exit O_{i+1} after O_i. Unless O_{i+1} intersects the line-stabbing wedges of both definitions 1 and 2, there are no stabbing lines of the first $i+1$ disks.

First, discard portions of O_i that lie in O_{i+1} and update the support hulls and limiting lines as under definition 2 if the remaining portion of O_i no longer intersects one of the limiting lines. Next, discard portions of O_{i+1} that lie in O_i and update according to definition 1 if necessary.

If disks O_1 and O_2 intersect, then the initial support hulls A and B are the upper and lower intersection points, respectively, of the boundaries of O_1 and O_2. The initial limiting lines are the two orientations of the line through the two intersection points. □

Proof for Def. 4: The fourth definition is different from the others in that it involves choosing points rather than defining an order for intervals. There is an equivalent formulation in terms of intervals, however: no later interval may end before an earlier one begins.

Lemma 8 *Let $[a_i, b_i]$, for $i \in [1 \ldots n]$, be non-empty intervals of the real line. One can choose a set of points $\{p_1, p_2, \ldots, p_n\}$ with $p_i \in [a_i, b_i]$ and $p_i \le p_j$ for all $1 \le i < j \le n$ if and only if there is no pair $j < k$ with $b_k < a_j$. Furthermore, the p_is can be chosen from the set $\{a_1, a_2, \ldots, a_n\}$.*

Proof: Form a set of truncated intervals $[a'_i, b'_i]$ with $a'_i = \max_{j \le i} a_j$ and $b'_i = \min_{k \ge i} b_k$. If these intervals are non-empty then the set $\{a'_1, a'_2, \ldots, a'_n\}$ satisfies the lemma. Otherwise, some interval $[a'_i, b'_i]$ is empty; there is a $j \le i$ and a $k \ge i$ such that $b_k < a_j$. □

We are not able to give a linear time algorithm for this definition of visiting order because the line-stabbing wedge has non-constant complexity. When our disks are constant size polygons or equal radius circles, however, we can maintain the wedge by an intersection algorithm that allows us to stab i disks in $O(i \log i)$ time.

As before, we want the line-stabbing wedge of the first i disks to be the locus of all points p that have a line that visits the i disks before visiting p. Assume that we have two limiting lines t and t' that define an angle of less than 180° and let W_j be the region between these lines and not left of disk O_j. Define the line-stabbing wedge as the intersection $\bigcap_{j \leq i} W_j$, drawn shaded in figure 20.

We can maintain the wedge as n disks are added incrementally using $O(n \log n)$ total time, according the the following lemma.

Lemma 9 *One can incrementally form all wedges for a sequence of n convex polygons with $O(n)$ sides altogether or n unit radius circles in a total of $O(n \log n)$ time.*

Proof: A convex polygon is the intersection of the halfplanes defined by its sides, so it is sufficient to compute halfplane intersections incrementally. This can be done by the dual of Preparata's convex hull algorithm[36]: Store the edges of the current wedge in a binary search tree. To add a halfplane h, compute the intersection of h and the current wedge in $O(\log n)$ time and discard edges outside of h in $O(\log n)$ time apiece.

For equal radius circles, Melkman and O'Rourke[31] have shown that, when looking from the intersection point $t \cap t'$, the order of the centers of the circles is the reverse of the order of the edges bounding the wedge. By storing the centers in a binary search tree, they show how to update the wedge boundary in $O(n \log n)$ total time. □

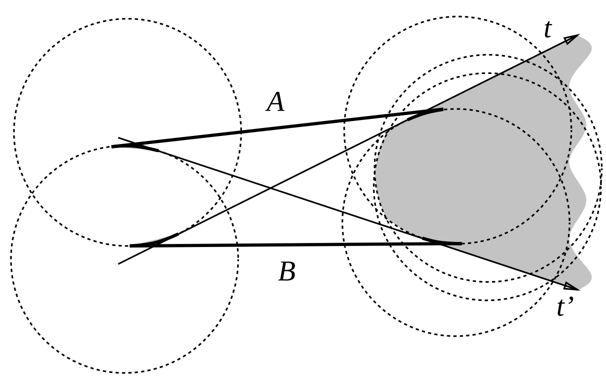

Figure 20: Wedge maintenance, definition 4

Let us first discuss updating the line-stabbing wedge and the description of stabbers when disk O_{i+1} is added. We'll discuss their initialization afterwards.

To begin, we must determine if O_{i+1} intersects the wedge—if it does not, then there is no ordered stabber of the first $i+1$ disks. We discard portions of O_{i+1} that lie outside the wedge. If what remains does not intersect the ccw (or cw) limiting line, then we must update the support hulls, limiting lines, and line-stabbing wedge. To perform the intersection, find the tangents from $t \cap t'$ to O_{i+1} (or, if $t \cap t'$ is inside O_{i+1}, use the rays along t and t') and break O_{i+1} into left and right portions where they hit its boundary. Form the region bounded by the right portion and the segments to $t \cap t'$, then intersect it with the wedge by walking along the wedge from t and t'—any edges walked on will be removed from the wedge. Then update the limiting lines and support hulls by Graham scan as under previous definitions. Finally, use the procedure of lemma 9 to update the wedge using the left portion of O_{i+1}.

To initialize the description of stabbers and the line-stabbing wedge, we begin by computing the intersection $\bigcap_{j<i} O_j$ incrementally by a procedure similar to that of lemma 9. While this intersection is non-empty, any line that stabs it stabs the disks in order according to definition 4. When O_i is disjoint from this intersection, then O_i must be disjoint from a disk O_j with $j < i$ and a slight modification of the intersection procedure of Melkman and O'Rourke[31] will give the disk O_j. We can then limit the directions of stabbers to lie between the directions of the inner common tangents directed from O_j to O_i and restart processing with O_1. The algorithm will find a line stabbing at least the first i disks. □

This completes the proof of theorem 7. In the next section we show that these stabbing line methods can be used to give linear-time algorithms to compute a stabbing chain that has at most twice the minimum number of links.

4.3. Ordered stabbing with a polygonal chain

The problem of ordered stabbing with a polygonal chain instead of a line brings its own complications. In this section, we extend the definitions of visiting order and look at restrictions that can be placed on the vertices of the chains. We note relationships to the Fréchet metric for curve similarity and line stabbing.

Figure 21 shows an example of a path π stabbing three disjoint objects O_1, O_2, and O_3. For each pair of objects, we can choose intervals of their intersections πs that have the correct order, but can hardly call π an ordered stabber of O_1, O_2, and O_3. Instead, we require that there is a sequence of intervals $I_1, I_2, \ldots, I_n$ in order along the path π such that I_j is a maximal connected interval of the intersection $\pi \cap O_j$. If these intervals happen to intersect, then we also apply our favorite definition 1, 2, 3 or 4.

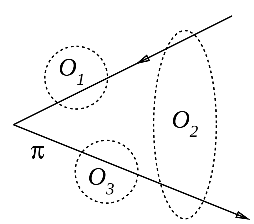

Figure 21: Pairwise order is not enough

One benefit of our chosen definition is that the stabbing problem can be viewed as minimum-link path problem in a non-manifold space $\mathcal{M}$. For $1 \le i \le n-1$, take a manifold M_i that is a Euclidean plane containing copies of objects O_i and O_{i+1}. Then identify (glue) the corresponding points in the copies of O_{i+1} contained in M_i and M_{i+1}. Any path in $\mathcal{M}$ from O_1 in M_1 to O_n in M_{n-1} visits the objects in order.

Further variations arise from different restrictions on the vertices of the approximation. We concentrate on three, listed in order of increasing restriction.

1. No restriction: The approximate path can turn anywhere.
2. Turn in tubes: Each vertex of the approximation must lie within a region bounded by two consecutive objects and their outer common tangents.
3. Turn in objects: Each vertex of the approximation must lie in one of the original objects.

The non-manifold space $\mathcal{M}$ constructed above can be modified so that any path in $\mathcal{M}$ automatically satisfies the second restriction: simply let M_i be the convex hull of O_i and O_{i+1} rather than the entire plane. The third restriction is of a different character.

If we combine the restriction that vertices must lie in tubes with the definition 4 for visiting order, then we obtain minimum link approximations under the *Fréchet metric*[3,14]. Two curves are within distance ε under this metric iff they have monotone parameterizations α and β, which are functions from $[0,1]$ to $\mathcal{R}^2$, such that $d(\alpha(t),\beta(t)) \leq \varepsilon$ for all $t \in [0,1]$. This can be understood intuitively as a person on α can walk a dog along β with a leash of length ε. The next theorem was suggested by Michael Godau (personal communication) and has been reported for the L_1 and L_∞ cases by Natarajan and Ruppert.[33]

Theorem 10 *Let $O_1, O_2, \ldots, O_n$ be a sequence of ε-balls and $c_1, c_2, \ldots, c_n$ be their centers. A minimum link chain stabbing $O_1, O_2, \ldots, O_n$ in order according to definition 4, whose vertices are constrained to lie in tubes, is a minimum link path with Fréchet distance at most ε from the polygonal chain $c_1, c_2, \ldots, c_n$.*

Proof: Let $\alpha : [0,1] \to \mathcal{R}^2$ be a parameterization of the polygonal chain $c_1, c_2, \ldots, c_n$ and let t_i be a parameter at which $\alpha(t_i) = c_i$.

For any curve β with Fréchet distance at most ε from α, the point $\beta(t_i) \in O_i$. By monotonicity of the parameterization, the sequence of points $\beta(t_1) \prec \beta(t_2) \prec \cdots \prec \beta(t_n)$ reveals that β visits the objects in order according to definition 4.

For any piecewise-linear curve β, let t be a parameter of one of its vertices and suppose that $t_i \leq t < t_{i+1}$. Then, in between visiting O_i and O_{i+1}, the curve β remains within ε of the line segment $\overline{c_i c_{i+1}}$, which is simply remaining in the convex hull of O_i and O_{i+1}. Thus the vertices of β lie in tubes.

Therefore, the minimum-link curve β with Fréchet distance at most ε from α is an ordered stabber satisfying the hypothesis. $\Box$

Using an algorithm for ordered stabbing with a line, there is a simple method to find a stabbing path for the strongest restriction using at most twice the minimum number of links.

Theorem 11 *One can compute an ordered stabbing path with vertices inside objects $O_1, O_2, \ldots, O_n$ that has less than twice as many segments as the minimum link stabbing path.*

Proof: Compute a line that stabs as many objects in order as possible. Then crop the line to a segment from the first to last objects stabbed, discard these objects and repeat. When all the objects have been stabbed, join the k segments formed into a path by adding $k-1$ segments.

Since each of the k segments, except for the last, stabs as many objects as possible, the minimum link path has at least k edges even if vertex placement is unrestricted. Therefore, the path constructed has less than twice as many edges as the minimum path. □

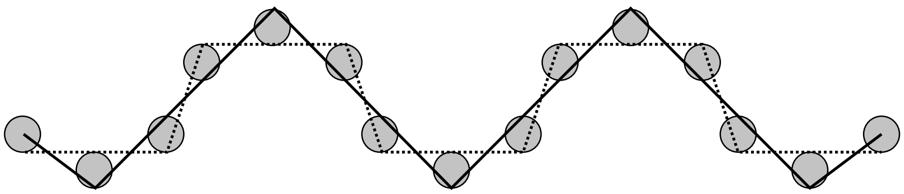

Figure 22: The greedy path (dotted) versus the minimum path (solid)

Figure 22 illustrates that when path vertices must lie inside stabbed objects, a greedy approach that always attempts to stab as many objects as possible can attain $2k-1$ links when the minimum link path has k links. The bound of theorem 11 is tight. This is in contrast to the algorithms for minimum link paths in simple polygons[13,21,39], where greedy methods do obtain a minimum link stabbing path.

4.4. A dynamic programming approach

In this section we develop a dynamic programming algorithm to stab intersecting unit disks with a minimum link chain. For each disk O_i, we compute a chain-stabbing wedge, defined below, and the length of the minimum link ordered stabbing chains that stab disks O_1 through O_i. For visiting orders 1 and 2, we obtain a minimum link stabbing paths in $O(n^2)$ time and linear space. For definition 4, the time increases to $O(n^2 \log n)$. Vertices must either be unrestricted or restricted to lie in tubes.

This should be compared to the general graph-based approach of Imai and Iri[24], which, in our terminology, creates a graph with an edge (j,k) if there is an ordered stabber from O_j through O_k and then search the graph for the shortest path. Our dynamic programming method shares the problem of a super-quadratic running time, but saves a factor of $O(n)$ in space by better organization of computation and relaxing the restriction that verticies be original data points. (We recently learned that Chin and Chan, in an unpublished manuscript, have improved Imai and Iri's algorithm to quadratic time.)

Extending the definition of wedges is key. In sections 4.1 and 4.2 we formed line-stabbing wedges under visiting orders 1, 2, and 4. For polygonal chains we define the *chain-stabbing wedge* W_i of the first i disks as the locus of all points p such that there is a minimum link chain visiting the first i disks and then p.

As an example, if disks O_1 through O_i can be stabbed by a line, then the chain-stabbing wedge W_i is a line-stabbing wedge, as defined in the previous sections. If the minimum path stabbing O_1 through O_i has $k > 1$ links, then wedge W_i is the union of line-stabbing wedges that first stab a point of a chain-stabbing wedge W_j that has a path of $k-1$ links and then stab disks O_{j+1} through O_i. This is not quite correct as stated, because we have not taken into account the restriction placed on turns. The true computation of W_i goes as follows. Let R_j be the region where the turn vertex between O_j and O_{j+1} can lie. Region R_j depends upon which of the three restrictions is placed on turns: With no restriction, R_j is the entire plane. For tubes, R_j is the region bounded by disks O_j and O_{j+1} and their outer common tangents. For each $j < i$ such that the chain-stabbing wedge W_j is formed by stabbing paths with $k-1$ links, compute the stabbing wedge for lines that stab, in order, $W_j \cap R_j$, O_{j+1}, O_{j+2}, $\ldots$, O_i. The union of these stabbing wedges is the chain-stabbing wedge W_i.

We show that chain-stabbing wedges can enlarge only when the path gains an extra link.

Lemma 12 *If the chain-stabbing wedges W_i and W_{i+1} both have minimum stabbing paths with k-links, then $W_{i+1} \subseteq W_i$.*

Proof: A point p is in W_{i+1} because there is a k-link path that visits the first $i+1$ disks before reaching p. The same path certifies that p is also in W_i. □

Next, we show that chain-stabbing wedges really are wedge-like.

Lemma 13 *The chain-stabbing wedge W_i is bounded by two rays and, depending on the definition of visiting order, a concave (def. 1) or convex (def. 2) portion of the boundary of O_i or a convex chain (def. 4) of the boundary of the intersection of O_h, O_{h+1}, $\ldots$, O_{i-1}, O_i for some $h \leq i$.*

Proof: We prove this by induction on the number of links in the chains forming chain-stabbing wedge W_i. Clearly, the lemma is satisfied by the stabbing wedges for lines, which are chain-stabbing wedges defined by 1-link chains.

Suppose that W_i is formed by m-link chains.

Let ω_j, for $j \leq i$, denote the line stabbing wedge for objects O_i, O_{i-1}, $\ldots$ down to O_j. (Since the ordering is reversed the definition of visiting order for ω_j should be the opposite of that of W_i. That is, substitute 1 for 2 and vice versa. Def. 4 is its own inverse.) Wedge W_j contributes rays to W_i if and only if the intersection $W_j \cap R_j \cap \omega_{j+1}$ is non-empty.

Suppose that W_j does contribute rays to W_i. We show that either no wedge W_k contributes to W_i, for all $k < j$, or else for the greatest $k < j$ whose wedge contributes, there is a line that stabs $W_{j-1} \cap R_k, O_{k+1}, \ldots, O_i$ and $W_j \cap R_j, O_{j+1}, \ldots, O_i$. This shows that the union of the stabbing wedges that make up W_i has at most two rays on the boundary.

So, consider a j such that $W_j \cap R_j \cap \omega_{j+1}$ is non-empty. If ω_{j+1} intersects $O_j \cap W_j$ then $W_{j-1} \cap R_{j-1} \cap \omega j$ is also non-empty and there is a desired ray starting in $O_j \cap W_j \cap \omega_j$. If ω_{j+1} does not intersect O_j then ω_{k+1} is empty for all $k < j$ and no wedge W_k contributes to W_i. What remains are the cases in which ω_{j+1} intersects O_j outside of W_j.

Suppose that ω_{j+1} intersects O_j after W_j. Since W_{j-1} does not intersect $O_j \cap \omega_{j+1}$, the intersection $W_{j-1} \cap \omega_j$ is empty in this case. Furthermore, since all objects O_k for $k < j$ will either miss ω_{k+1} or interesct after W_j, we can show that no rays are contributed to W_i from any such W_k.

Finally, suppose that ω_{j+1} intersects O_j before W_j. Again, W_{j-1} does not intersect $O_j \cap \omega_{j+1}$. Now there is an intersection point of the boundaries of W_{j-1} and ω_j that is either in R_{j-1} or outside of R_{j-1}. If it is in R_{j-1}, then some ray from that point is the desired ray. If not, then there is no contribution from W_{j-1} and will be no contribution until that point is in the next R_k. This establishes the lemma. □

We now sketch the dynamic programming algorithm.

Theorem 14 *Under visiting order definitions 1, 2, or 4, one can compute the minimum link path visiting disks $O_1, O_2, \ldots, O_n$ in order that either has no restrictions on vertices or has vertices in the convex hull of consecutive disks. Space is $O(n)$. Under definitions 1 and 2 the time is $O(n^2 \log n)$. Under definition 4, the time increases to $O(n^2 \log^2 n)$.*

Proof: For definitions 1 and 2 (entering or leaving the objects in the specified order) lemma 13 says that a chain-stabbing wedge is an object of constant complexity—we can store all chain-stabbing wedges in $O(n)$ space. We also store the number of links to each chain-stabbing wedge.

With this information, we can carry out the computation of an m-link chain-stabbing wedge W_i described above: given the descriptions of all $(m-1)$-link chain-stabbing wedges $W_j, W_{j+1}, \ldots, W_k$, we compute each of the line-stabbing wedges of the objects $W_\ell \cap R_\ell$, $O_{\ell+1}$, O_{j+2}, $\ldots$, O_i, for $j \le \ell \le k$, where R_ℓ is the region where the turn vertex between O_ℓ and $O_{\ell+1}$ can lie; R_ℓ is convex and has a constant-size description for the restrictions we allow.

This computation can be carried out in $O(n \log n)$ time by initially running the line-stabbing algorithm of section 4.2 on the objects ordered from O_i down to O_{k+1}—this requires reversing the current definition of visiting order. Then, looping from $\ell = k$ down to $\ell = j$, compute the limiting lines that would be formed by adding object $W_\ell \cap R_\ell$: one can do this in logarithmic time by binary search of the current support hulls. Next, insert O_ℓ into the current line stabbing wedge and decrement ℓ. The limiting lines computed by binary search and O_i delimit the desired line-stabbing wedges.

The union of these stabbing wedges can be computed by finding the extreme rays. Since the computation of a single wedge $O(n \log n)$, the total time is bounded by $O(n^2 \log n)$.

For definition 4, we cannot store the chain-stabbing wedges because they have non-constant complexity. We store only the two bounding rays for chain-stabbing wedges and construct wedge boundaries when we need them by intersecting arcs of unit circles using Melkman and O'Rourke's algorithm[31] as in section 4.2. This, of course, further complicates the algorithm for finding the bounding rays.

To compute an m-link chain-stabbing wedge W_i, we find the range of all $(m-1)$-link chain-stabbing wedges $W_j, W_{j+1}, \ldots, W_k$. Then we compute line-stabbing wedges from O_i down to O_j and record all the changes to the support hull data structures so that we can delete the objects $O_j, \ldots, O_k$ by playing the record backwards. We compute the wedge W_j by intersecting the objects before O_j with the wedge defined by O_j and its two extreme rays, if necessary. Starting with $\ell = j$, we compute the limiting lines for $R_\ell \cap W_\ell$ and $O_{\ell+1}, \ldots, O_i$ by finding common tangents between $R_\ell \cap W_\ell$ and the support hulls of $O_{\ell+1}, \ldots, O_i$ with nested binary search. Then we intersect the boundary of $O_{\ell+1}$ with the boundary of the wedge W_ℓ, if necessary, delete object $O_{\ell+1}$ from the the current line stabbing wedge and decrement ℓ. The computation for a single wedge is $O(n \log^2 n)$, so the total time is $O(n^2 \log^2 n)$. □

4.5. A linear-time greedy algorithm

When consecutive objects are disjoint, then we can give a linear-time greedy algorithm that computes a minimum-link stabbing chain. As in the previous section, vertices are unrestricted or are restricted to lie in *tubes*—inside the convex hull of two consecutive objects.

Natarajan and Ruppert[33] have independently developed a similar algorithm for stabbing unit squares and have used it to compute minimum link L_1 and L_∞ approximations to polygonal chains when each original segment is longer than unity. They also noted the relationship to the Fréchet metric that we established in theorem 10.

The idea is the following. If we have a line stabbing wedge W for the first $i-1$ objects and find that O_i does not intersect W, then we must consider how many of the first $i-1$ objects that we should stab with the first segment. It may be advantageous not to stab them all, as figure 22 shows. What we find, however, is that we can take the posibly turning locations into account in the way that we initialize the constraints for the next stabbing line. If O_j is below the wedge W, for example, then for the upper constraint we want only that the line stabs W. We put in all the objects O_1 through O_{i-1} as lower constraints, however, because we do not want to miss one of the objects by passing underneath it. We content ourselves with a detailed sketch of the proof.

Theorem 15 *Let $\mathcal{O} = O_1, O_2, \ldots, O_n$ be a sequence of convex objects in which consecutive objects are disjoint. One can compute, in $O(n)$ time, the minimum-link ordered stabbing path whose vertices either have no restrictions or lie in or between consecutive objects.*

Proof: We begin by finding the longest prefix that can be stabbed by a line using algorithm 1. We record the current limiting lines after we add each new object. If the prefix has i objects, then the algorithm ends in $O(i)$ time with a stabbing wedge, which is bounded by two limiting lines and a portion of O_i and does not intersect O_{i+1}.

Let us first consider restricting the vertices to lie in *tubes,* that is, in the convex hull of consecutive objects. We consider three cases, illustrated in figure 23—these cases could actually be unified at the cost of making the exposition completely opaque. For each case, we consider first the computation when vertices lie in *tubes,* that is, in the convex hull of consecutive objects, and second the modifications required if the vertices are unrestricted.

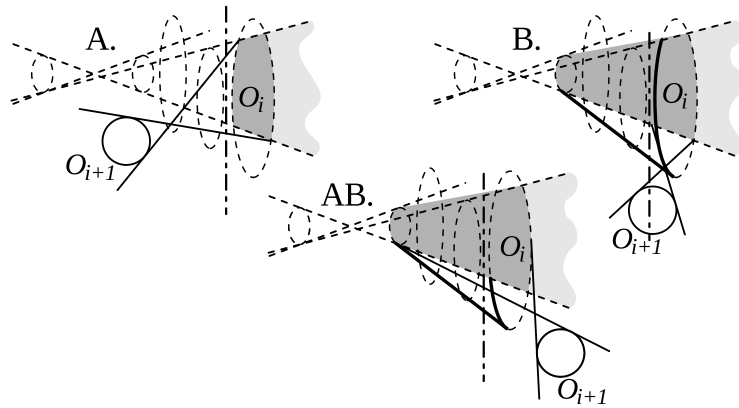

Figure 23: Cases A, B, and AB

Case A: Case A obtains when some line separates object O_i from objects O_{i-1} and O_{i+1}. A vertex of the approximation must lie between O_{i-1} and O_{i+1}, and if vertices are restricted to lie in tubes, then this vertex lies in the portion of O_i that lies in the stabbing wedge, shaded heavily in the figure. We can run algorithm 1 starting with this portion of O_i to find the next sequence that can be stabbed.

If the vertices are unrestricted, then we begin algorithm 1 with the stabbing wedge, shaded in the figure, which is an object that is disjoint from O_{i+1}. This beginning implicitly assumes that the vertex between O_{i-1} and O_{i+1} should occur in or after O_i. One can argue, however, that no possible stabbers are lost by this assumption: although removing O_i may enlarge the stabbing wedge, any segment of a minimum-link stabbing chain that originates from a point in the enlarged wedge must cross O_i before O_{i+1} and thus must cross the stabbing wedge bounded by O_i.

Case B: Case B obtains when some line separates O_{i-1} from O_i and O_{i+1}. Let us assume that O_{i+1} is right of the cw limiting line t' as shown in figure 23B. The computation when O_{i+1} is left of the ccw limiting line t is symmetric.

We begin algorithm 1 with above and below support hulls defined by different objects. For the above hull we use a single object, the convex hull of O_i and the point $t \cap t'$, if the vertices must lie in tubes (darker shading), or the wedge right of t and left of t' (dark and light shading), if vertices are unrestricted. For the below hull, we begin with the support hull of a sequence of objects: start from the second contact object O_j of the cw limiting line t' and continue through O_i—trim the top of each objects by the ccw limit line that existed when the object was inserted. This support hull is drawn darkly in figure 23B.

Decoupling the above and below constraints avoid the implicit committment to place a vertex between a given pair of consecutive objects that lead to extra segments in the algorithm of the previous section. As illustrated in figure 23B, the vertex can be placed on t' between two consecutive objects and the below support hull will ensure that objects after the vertex are stabbed by the next segment of the path. This choice of constraints captures the boundary of the illuminated region in the space $\mathcal{M}$.

Case AB: In case AB, the separators of O_{i-1} and O_i intersect O_{i+1}. Case AB is handled just like case B, with the decision whether O_{i+1} is left of t or right of t' based on the intersection of O_{i+1} with a separator of O_{i-1} and O_i. In figure 23, the initial above and below support hulls for B and AB are the same.

All cases can be set up in time proportional to the number of objects. Each object in the entire sequence is considered at most twice in the computation of a minimum-link stabber, therefore the total computation is linear. □

5. Conclusions and open problems

We have examined minimum link approximations that lie in convolutions or are ordered stabbers as part of a basic approach to approximating paths, polygons, and subdivisions. We have developed some efficient algorithms and indicated that others are unlikely to ever be developed.

There are many avenues that we hope to explore further—the most important being practical studies of implementations of theoretically efficient approximation methods. A few of the many open questions that remain are: Is computing the minimum link simple polygon enclosing all holes *NP*-complete? What other restrictions on approximation can be handled in subquadratic time? For example, the vertices may be required to lie within some $\delta < \varepsilon$ of the original path. Can subquadratic time algorithms be developed for ordered stabbing of intersection objects or for other definitions of visiting order?

Acknowledgements

We thank Michael Godau for suggesting theorem 10 and Jim Ruppert for a preliminary version of his paper with Natarajan.[33]

References

1. P. K. Agarwal, M. Sharir, and P. Shor. Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences. *Journal of Combinatorial Theory, Series A*, 52:228–274, 1989.
2. H. Alt, J. Blömer, M. Godau, and H. Wagener. Approximation of convex polygons. In *Seventeenth International Colloquium on Automata, Languages and Programming*, number 443 in Lecture Notes in Computer Science, pages 703–716. Springer-Verlag, 1990.
3. H. Alt and M. Godau. Measuring the resemblance of polygonal curves. In *Proceedings of the Eighth Annual ACM Symposium on Computational Geometry*, pages 102–109, 1992.
4. M. A. Armstrong. *Basic Topology.* McGraw-Hill, London, 1979.
5. R. Bellman. On the approximation of curves by line segments using dynamic programming. *Communications of the Association for Computing Machinery*, 4:284, 1961.

6. M. Blakemore. Generalisation and error in spatial data bases. *Cartographica*, 21:131–139, 1984.
7. B. Buttenfield. Treatment of the cartographic line. *Cartographica*, 22:1–26, 1985.
8. D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Implicitly searching convolutions and computing depth of collision. In *Algorithms: International Symposium Sigal 90*, number 450 in Lecture Notes in Computer Science, pages 165–180, 1990.
9. D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.
10. P. Egyed and R. Wenger. Ordered stabbing of pairwise disjoint convex sets in linear time. *Discrete Applied Mathematics*, 31:133–140, 1991.
11. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.
12. M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
13. S. K. Ghosh. Computing the visibility polygon from a convex set and related problems. *Journal of Algorithms*, 12:75–95, 1991.
14. M. Godau. *Die Fréchet-Metrik für Polygonzüge—Algorithmen zur Abstandsmessung und Approximation.* PhD thesis, Fachbereich Mathematik, FU Berlin, 1991.
15. R. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133, 1972.
16. L. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 100–111, 1983.
17. L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Minimum link approximation of polygons and subdivisions. In W. L. Hsu and R. C. T. Lee, editors, *ISA '91 Algorithms*, number 557 in Lecture Notes in Computer Science, pages 151–162. Springer-Verlag, 1991.
18. L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete & Computational Geometry*, 2:175–193, 1987.
19. S. L. Hakimi and E. F. Schmeichel. Fitting polygonal functions to a set of points in the plane. *CVGIP: Graphical Models and Image Processing*, 53(2):132–136, 1991.
20. J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplification algorithm. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 134–143. IGU Commision on GIS, 1992.
21. J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications*, 1993.
22. H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36:31–41, 1986.
23. H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162, 1986.
24. H. Imai and M. Iri. Polygonal approximations of a curve—formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology.* North Holland, 1988.
25. K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete & Computational Geometry*, 1:59–71, 1986.

26. D. Leven and M. Sharir. Planning a purely translational motion for a convex polygonal object in two dimensional space using generalized Voronoi diagrams. *Discrete & Computational Geometry*, 2:9–31, 1987.

27. D. Litchenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

28. R. B. McMaster. A statistical analysis of mathematical measures for linear simplification. *The American Cartographer*, 13:103–116, 1986.

29. R. B. McMaster. Automated line generalization. *Cartographica*, 24(2):74–111, 1987.

30. R. B. McMaster. The integration of simplification and smoothing algorithms in line generalization. *Cartographica*, 26(1):101–121, 1989.

31. A. Melkman and J. O'Rourke. On polygonal chain approximation. In G. T. Toussaint, editor, *Computational Morphology*. North Holland, 1988.

32. J. R. Munkres. *Topology: A First Course*. Prentice-Hall, Englewood Cliffs, N.J., 1975.

33. B. K. Natarajan and J. Ruppert. On sparse approximations of curves and functions. Manuscript, 1991.

34. J. O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Communications of the Association for Computing Machinery*, 24(9):574–578, Sept. 1981.

35. J. Perkal. On the length of empirical curves. In *Discussion Paper 10, Michigan Inter-University Community of Mathematical Geographers*, University of Michigan, Ann Arbor, 1966.

36. F. P. Preparata. An optimal real time algorithm for planar convex hulls. *Communications of the Association for Computing Machinery*, 22(7):402–405, 1979.

37. U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Vision, Graphics, and Image Processing*, 1:244–256, 1972.

38. A. Rosenfeld. Axial representation of shape. *Computer Vision, Graphics, and Image Processing*, 33:156–173, 1986.

39. S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35:99–110, 1986.

40. G. Toussaint. On the complexity of approximating polygonal curves in the plane. In *Proc. IASTED, International Symposium on Robotics and Automation*, Lugano, Switzerland, 1985.

41. E. R. White. Assessment of line-generalization algorithms using characteristic points. *The American Cartographer*, 12(1):17–27, 1985.

42. C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, 2:365–393, 1987.