Robust Implementation of a Convex Polygon Stabber for a Set of Isothetic Line Segments

Xin Wu   Xijie Zeng   Bryan St. Amour   Asish Mukhopadhyay *

July 14, 2009

Abstract

In this paper, we discuss the algorithm engineering aspects of a recently-reported $O(n^2)$-time algorithm [6] for computing a minimum-area convex polygon that intersects a set of n isothetic line segments.

1 Introduction

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n isothetic line segments in the plane. Let P be any convex polygon. A segment s_i intersects P if it lies in the interior of P or intersects its boundary. In [6] Mukhopadhyay et al proposed an $O(n^2)$ algorithm for computing such a polygon of minimum area, P_{min}.

Implementing this algorithm turned out to be quite challenging, with the final code running into nearly 10000 lines of JAVA code. In this paper we discuss the challenges we faced in course of the implementation and the strategies we devised to overcome these.

To make the paper self-contained we outline in the next section the salient features of the algorithm in [6], along with details of how each of the major steps was implemented. In the following section, we discuss our handling of the input and output. Results of some experiments are provided in the next section, and finally we conclude.

2 Algorithm outline

To devise an algorithm we need to understand what P_{min} looks like.

Four functions are associated with each line segment s - `top`(s), `bot`(s), `left`(s) and `right`(s) that respectively return the top, bottom, left and right end-points of s. For a vertical line segment, the functions `left`() and `right`() are undefined, while `top`() and `bot`() are undefined for a horizontal line segment.

Step1: Finding extreme segments

We first find four particular line segments in S.

- A "top-most" extreme segment, $\overline{tT}$: Find a vertical segment t_1 such that `bot`(t_1) has the maximum y-value, and the highest horizontal segment t_2. If `bot`(t_1) is above t_2, then t_1 is the "top-most" extreme segment. Otherwise, t_2 is the "top-most".
- A "right-most" extreme segment, $\overline{rR}$: Find a horizontal segment r_1 such that `left`(r_1) has the maximum x-value, and find the vertical segment r_2 with the largest x-value. If `left`(r_1) is to the right of r_2, then call r_1 the "right-most" extreme segment. Otherwise, r_2 is the "right-most".

*Corresponding author, School of C omputer Science, University of Windsor

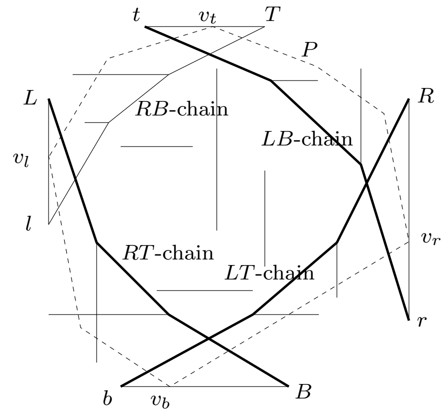


Figure 1: Four hull chains for isothetic line segments

- A "bottom-most" extreme segment, $\overline{bB}$: If the vertical segment b_1, for which $\mathtt{top}(b_1)$ has the minimum y-value, is completely below the lowest horizontal segment b_2, then b_1 will be the "bottom-most" extreme segment . Otherwise, b_2 is the "bottom-most".
- A "left-most" extreme segment, $\overline{lL}$: If the horizontal segment l_1, for which $\mathtt{right}(l_1)$ has the minimum x-value, is completely to the left of the left-most vertical segment l_2, then l_1 will be the "left-most" extreme segment. Otherwise, l_2 is the "left-most" one.

See Fig. 1 for an example. In an actual input, all of these extreme segments may not exist and even when they do there can be more than one extreme segemnt of a given type. We discuss later how our implementation deals with such degenerate input.

Step 2: Computing critical chains

Next we compute, using a standard convex hull algorithm, the following 4 convex chains.

- A convex chain going from $\overline{lL}$ to $\overline{tT}$ that is part of the convex hull of $\mathtt{right}(s)$ and $\mathtt{bot}(s)$ of all segments s in S, whenever these are defined. We call this the RB-chain.
- A convex chain going from $\overline{tT}$ to $\overline{rR}$ that is part of the convex hull of $\mathtt{left}(s)$ and $\mathtt{bot}(s)$ of all segments s in S, whenever these are defined. We call this the LB-chain.
- A convex chain going from $\overline{rR}$ to $\overline{bB}$ that is part of the convex hull of $\mathtt{left}(s)$ and $\mathtt{top}(s)$ of all segments s in S, whenever these are defined. We call this the LT-chain.
- A convex chain going from $\overline{bB}$ to $\overline{lL}$ that is part of the convex hull of $\mathtt{right}(s)$ and $\mathtt{top}(s)$ of all segments s in S, whenever these are defined. We call this the RT-chain.

These chains (see Fig. 1 for an example set of segments) help us formulate a structural characterization of convex polygons that intersect all the segments in S as in the lemma below.

Lemma 1 *Let P be any convex polygon that intersects all the line segments in S. Then the upper-left convex chain of P must be on or above and to the left of the RB-chain; the upper-right chain of P must be on or above and to the right of the LB-chain; the lower-right chain of P must be on or below and to the right of the LT-chain, and; the lower-left chain of P must be on or below and to the left of the RT-chain.*

Proof: See [6]. □

Step 3: Computing P_{min}

It is possible that for a given set of input segments one or more of these chains may be missing. More about this later.

Thus, the class of convex polygons P under consideration consist of *at most* four convex chains, each of which joins a pair of extreme vertices (v_l, v_t, v_r, and v_b) that lie respectively on the extreme segments $\overline{lL}$, $\overline{rR}$, $\overline{tT}$, and $\overline{bB}$.

We can be more precise about the location of these extreme vertices. For example, v_l will lie in one of the subintervals of $\overline{lL}$, obtained by extending the edges of the critical chains to partition this extreme segment. The points in a given subinterval form an equivalence class in the sense that the edges of an intersecting convex polygon P incident on a point of this subinterval are tangent to the critical chains at the same points. In Figure 8, for example, extreme segment L is partitioned by points A and B, hence L has three intervals.

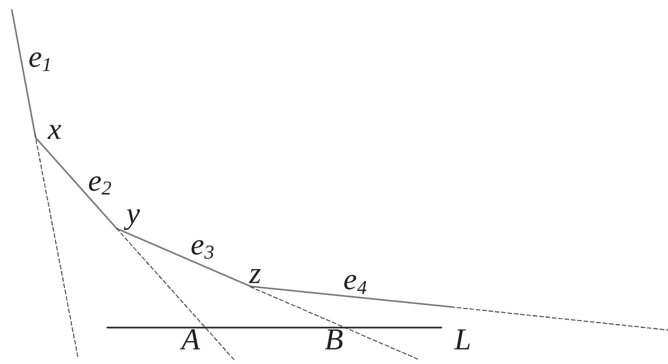


Figure 2: Partitioning an extreme segment into subintervals

Sub-interval generation was implemented by the method setIntervalsBy(MPCriticalChain) of class MPExtremeSegment. The pseudo-code for this is shown below.

Algorithm setIntervalsBy ($MPCriticalChain$)

1. Find the first edge e of critical chain which can partition extreme segment.
2. Initiate the tangent point of each interval to the first endpoint of e.
3. For each edge E of critical chain after e:

    3.1. Get the intersection point P by extending edge E.

    3.2. If P is null, continue.

    3.3. For each interval I on extreme segment:

        3.3.1. If P is located on interval I:

            3.3.1.1. If P is identical with the first endpoint of interval I, set tangent point of I to the second endpoint of E

            3.3.1.2. Else:

                3.3.1.2.1. Split interval I into two parts, $I1$ and $I2$.

                3.3.1.2.2. Set tangent point of $I2$ to the second endpoint of E

            3.3.1.3. Set each tangent point of intervals behind $I2$ to the second endpoint of E

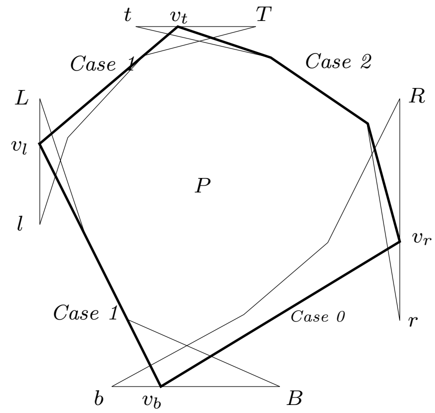


Figure 3: An example of each connection

An input requirement of the above method is that all edges of critical chain and the two endpoints of any edge be given in counterclockwise order. For the example in Fig. 8, this method will set x as the tangent point of the left interval on L,y as the tangent point of the middle interval, and z is the tangent point of the right interval.

In addition to generating intervals by extending edges of critical chains, intervals could also be created by the cutting of other extreme segments. Experiments show that this helps in finding a minimum polygon. The method setIntervalsBy(MPExtremeSegment) take charge of this task.

A convex chain of P_{min}, joining two extreme vertices and referred to as a *connection* here and after, is one of the following types (see Figure 3).

Case 0 A single edge that does not touch any of the critical chains.

Case 1 A single edge that is tangent to an underlying critical chain.

Case 2 A convex chain composed of many edges, in which case an underlying critical chain contributes some structure to it.

Two extreme vertices lying on a pair of vertical and horizontal extreme segments are considered to be *adjacent*; otherwise they are *non-adjacent*. Each of the above types of connections can be further subdivided according to which pairs of extreme vertices they join. These are:

(i) A connection joining two adjacent extreme vertices (for example: v_l to v_t; v_t to v_r; etc.; see Figure 3).

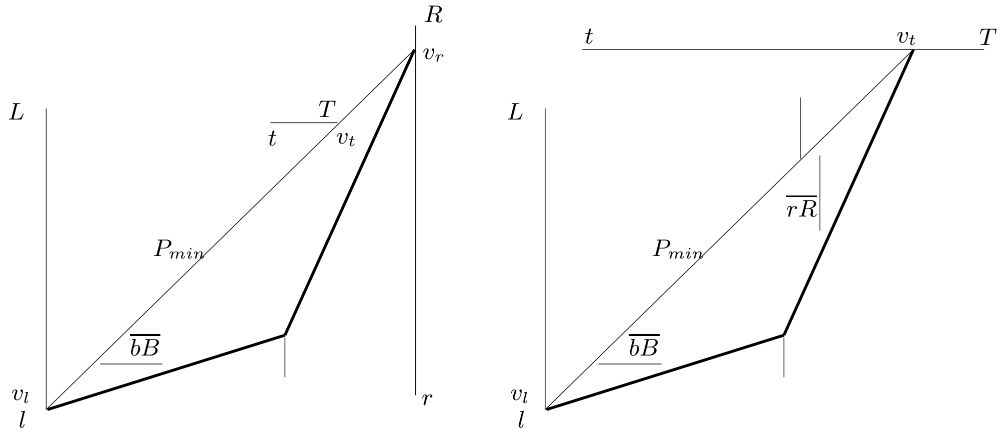
(ii) A connection joining two non-adjacent extreme vertices (v_l to v_r or v_t to v_b), bypassing one of the extreme segments. As a result, this extreme segment is forced to lie in the interior of P_{min}. See Figure 4(a), in which v_t is not actually a vertex of P_{min}, and in which v_b does not exist.

(iii) There can be two connections that join two adjacent extreme vertices: one of (i) and another that bypasses the other extreme segments. As a result, these extreme segments are forced to lie in the interior of P_{min}. See Figure 4(b).

However, it is impossible for a connection to bypass three extreme segments. For a proof see [6].

In [6] it was shown that to compute P_{min} we have to consider 219 different configurations of connections, calculated as follows:

- 81 different possibilities when there are four connections,



(a) The connection in bold bypasses $\overline{bB}$ (b) The connection in bold bypasses $\overline{rR}$ and $\overline{bB}$

- 108 possibilities when there are three connections (3^3 times four, for each of the four extreme segments that can be bypassed),
- 20 possibilities when we have one regular connection and one that bypasses two extreme segments (four, for each position of the regular connection, times five, since the connections can be 0-2, 1-2, 2-0, 2-1, or 2-2), and
- 10 possibilities when we have two connections each bypassing one extreme segment (two possibilities for the bypassed segments, times five, since the connections can be 0-2, 1-2, 2-0, 2-1, or 2-2).

During implementation, we observed that this number can be reduced to 47 only by considering isomorphism and symmetry among these different configurations. The details are as follows:

We use the notation $x_1x_2x_3x_4$ to denote a four connection configuration where $x_i (i = 1, 2, 3, 4)$ is one of the three connection types: 0, 1 or 2; x_1 connects $\overline{tT}$ and $\overline{lL}$; x_2 connects $\overline{lL}$ and $\overline{bB}$; x_3 connects $\overline{bB}$ and $\overline{rR}$; x_4 connects $\overline{rR}$ and $\overline{tT}$.

In terms of this notation, $x_1x_2x_3x_4, x_2x_3x_4x_1, x_3x_4x_1x_2$ and $x_4x_1x_2x_3$ are all isomorphic configurations that can be handled by the same procedure by permuting the input parameters. For example, if the procedure *Comp1012*($\overline{tT}$, $\overline{lL}$, $\overline{bB}$, $\overline{rR}$, *RB, RT, LT, LB)* computes the minimum area polygon for the configuration 1012, then *Comp1012*($\overline{lL}$, $\overline{bB}$, $\overline{rR}$, $\overline{tT}$, *RT, LT, LB, RB)* will be invoked for the configuration 0121. Similarly, by a proper ordering of the parameters, *Comp1012* can also compute minimum area polygon for the configurations 1210 and 2101.

If we assume that the inputs are in the first quadrant of an orthogonal xy-coordinate system, we will consider the configurations $x_1x_2x_3x_4$ and $x_4x_3x_2x_1$ as mirror images of each other with respect to the y-axis. We will say such configurations are symmetric; they can be handled by the same procedure. Suppose the procedure *Comp0012* computes the minimum area polygon for the configuration 0012. If we mirror all the inputs with respect to the y-axis, call *Comp0012* on the mirrored inputs and reflect the output in the y-axis, we will have solved the problem for the configuration 2100 on the original input segments.

We do not have to consider symmetries of $x_1x_2x_3x_4$ with respect to the x-axis or the origin. This is because $x_2x_1x_4x_3$ (x-axis symmetric configuration) and $x_4x_3x_2x_1$ are isomorphic configurations, and so are $x_3x_4x_1x_2$ (origin symmetric) and $x_1x_2x_3x_4$.

By using these two optimization strategies, we get the following 47 non-isomorphic, non-symmetrical configurations.

- For all 4-connection configurations, there are 24 non-isomorphic ones in Table 1. However, 0012 and 1200 are isomorphic pairs, 1200 and 0021 are symmetrical pairs. Thus, 0012 and 0021 can be handled by the same procedure. The same reason applies to 0122 and 0221, 1120 and 1102. Consequently, there are 21 non-isomorphic, non-symmetrical 4-connection configurations.

Table 1: Non-isomorphic 4-connection configurations

0000	1111	2222	0111	0222	1222
1100	2200	1122	0101	0202	1212
0001	0002	1112	0012	1012	0122
1020	1120	1202	0021	1102	0221

- For all 3-connection configurations, there are 27 non-isomorphic ones in Table 2. (3 connections and 3 types for each connection; There is no need to multiply by four, because of isomorphism.) Among these 27 non-isomorphic configurations, there are 18 non-symmetrical ones in Table 3. Note that for a given triple xyz, it could be x, y, or z that bypasses a segment. That is why there appear to be duplicates in the table; they are necessary.

Table 2: Non-isomorphic 3-connection configurations

000	010	020	001	011	021
002	012	022	100	110	120
101	111	121	102	112	122
200	210	220	201	211	221
202	212	222			

Table 3: Non-isomorphic non-symmetric 3-connection configurations

000	010	020	001	011	021
002	012	022	101	111	121
102	112	122	202	212	222

- For all 2-connection (one regular, one bypasses two extreme segments) configurations, there are 5 non-isomorphic, non-symmetrical ones 02, 12, 20, 21, 22. For configuration 02, Case 0 is a regular connection while Case 2 bypasses two extreme segments. For configuration 20, Case 2 is regular, while Case 0 bypasses two. So 02 and 20 are a non-isomorphic, non-symmetrical pair. The same reason applies to 12 and 21.
- For all 2-connection (both bypass one extreme segment) configurations, there are 3 non-isomorphic, non-symmetrical ones 02, 12 and 22.

When considering a potential solution polygon consisting of at least three connections, we can discern one or more of the following patterns in the configuration of connections.

Pattern A Two occurrences of Case 2: This divides the problem into two *independent* sub-problems. If the two connections occur on "adjacent" critical chains (like RB and LB, or LB and LT, etc.) then the problem is reduced to searching through $O(n)$ intervals on one extreme segment, and searching through $O(n^3)$ interval triplets, for the other three extreme segments. See Figure 4 for an example of this.

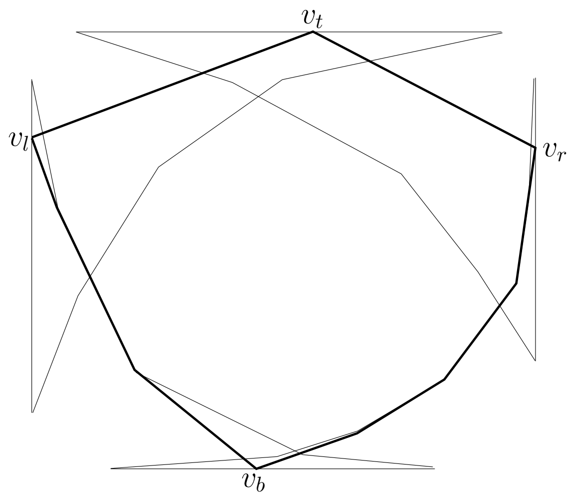


Figure 4: Pattern A (lower half of bold polygon) and Pattern C (upper half of bold polygon)

If the connections occur on "opposite" chains (RB and LT, or RT and LB) then the problem is reduced to choosing from $O(n^2)$ interval pairs for the two extreme segments on one side of the chains, and choosing from $O(n^2)$ interval pairs for the two extreme segments on the other side.

Pattern B An occurrence of Case 1: (For example, the connection between v_l and v_t in Figure 3.) There are only $O(n)$ interval pairs that are connected by a line that is tangent to the underlying critical chain. One can think of a tangent line rotating along the underlying critical chain: this line will hit only $O(n)$ interval pairs. So, there will be $O(n^3)$ interval quadruplets to consider.

Pattern C Two adjacent occurrences of Case 0: (See Figure 4) The extreme segment attached to these connections will not have to be divided into any intervals, since the underlying critical chains will not contribute any structure to that part of P_{min}. Again, there will be only $O(n^3)$ interval quadruplets to consider.

These patterns can be used to show that only $O(n^2)$ different polygons have to considered to determine the minimum (see [6] for details).

Determination of the types of connections was an important part of our implementation. We describe below how we went about this.

The essence of this algorithm is to generate convex polygons by combining every possible connection between extreme segments. It will search 47 possible configurations of connections in total. This implies the necessity of determining whether a specific connection is Case 0, or Case 1, or Case 2, which are defined in section **??** and illustrated in Figure 3.

Some involved functions are characterized as follows.

isCase0
This function is used to check whether a connection between two adjacent extreme segments has the type of Case 0. It returns true as long as such a connection has no intersection or overlapping with the corresponding critical chain between same extreme segments.

isCase0CommEndpoint
According to the discussion of Pattern C, we know that when two adjacent connections are Case 0, the common vertex of the two connections must be one of the endpoints of the corresponding extreme segment.

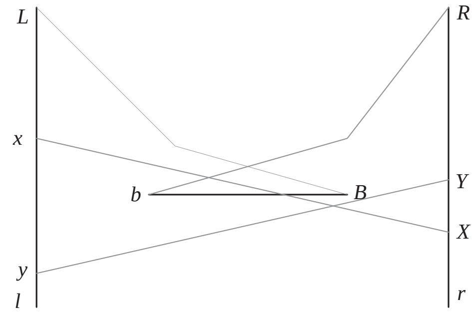


Figure 5: Two connections are Case 0 bypass

This also means that the common vertex is on a critical chain. This function is similar to isCase0, but it allows one vertex of the connection to be on a critical chain instead of totally no touching.

isCase0Bypass
A connection can join two non-adjacent extreme vertices, hence it bypasses one of the extreme segments. Consequently, this extreme segment is forced to lie in the interior of the minimal convex polygon. Though there is no accurate definition of Case 0 in bypass situation, this task can be performed in a somewhat tricky way.

In our implementation, a connection is considered having a type of Case 0 if

1. it touches the bypassed extreme segment, or it makes such extreme segment contained in the convex polygon; and
2. it is away from involved critical chains with the only exception that intersecting the nearest edge of critical chain.

The above two conditions make sure that the connection will not cut through the core area and, at the same time, the candidate polygon will intersect the bypassed extreme segment.

Two examples of Case 0 connection are illustrated in Figure 5. Connection $\overline{xX}$ intersects extreme segment $\overline{bB}$, and does not cut through the core area which is partially bounded by the RT-chain and LT-chain. Undoubtedly, connection $\overline{xX}$ has the type of Case 0 bypass. On the other hand, although connection $\overline{yY}$ does not intersect $\overline{bB}$, it is still capable of including $\overline{bB}$ in the polygon, to which $\overline{yY}$ contributes itself as an edge. The consequential question is how to check the inclusion. Assuming $\overline{yY}$ is a directed line segment and follows the counterclockwi se order, we can find that $\overline{bB}$ is included in the polygon if it is located on the left side of $\overline{yY}$. This is the exact approach that we implement in the program.

isCase0BypassCommEndpoint
This function is used in the Pattern C when one of the two adjacent Case 0 connections is trying to bypass any extreme seg ment. It is similar with isCase0Bypass but allows a common endpoint.

isCase1Bypass
This function takes three points to represent a connection, that is, two endpoints and an in-between point at which the connection is tangential to the critical chain. Only by doing this we can make sure that, in a bypass situation, no extreme segment will be missed when generating polygons.

isCase2Bypass
This is similar to isCase1Bypass but it deals with the configuration illustrated in Figure 6.
Here's a summary of the algorithm

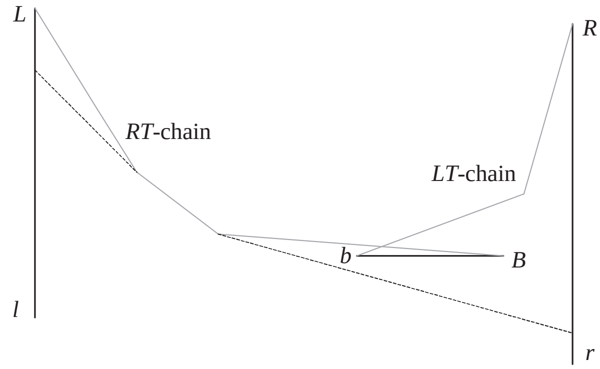


Figure 6: Case 2 bypass

Algorithm IsotheticMinPolyStabber(S)

1. Compute the critical chains RB, LB, LT, and RT.
2. Extend the edges of these chains to partition the extreme segments $\overline{lL}$, $\overline{tT}$, $\overline{rR}$, and $\overline{bB}$; store the extended edges and corresponding points of tangency, on the critical chains, for each partition point.
3. For each configuration of connections:
   3.1. For each possible tuple of intervals:
      3.1.1. Solve an optimization problem with suitable constraints, resulting in two to four extreme vertices.
      3.1.2. Find the area of the polygon using these extreme vertices.
      3.1.3. If this is the smallest polygon seen so far then store these extreme vertices as the optimal ones.
4. Report the minimum polygon by joining the optimal extreme vertices, using their points of tangency to the critical chains.

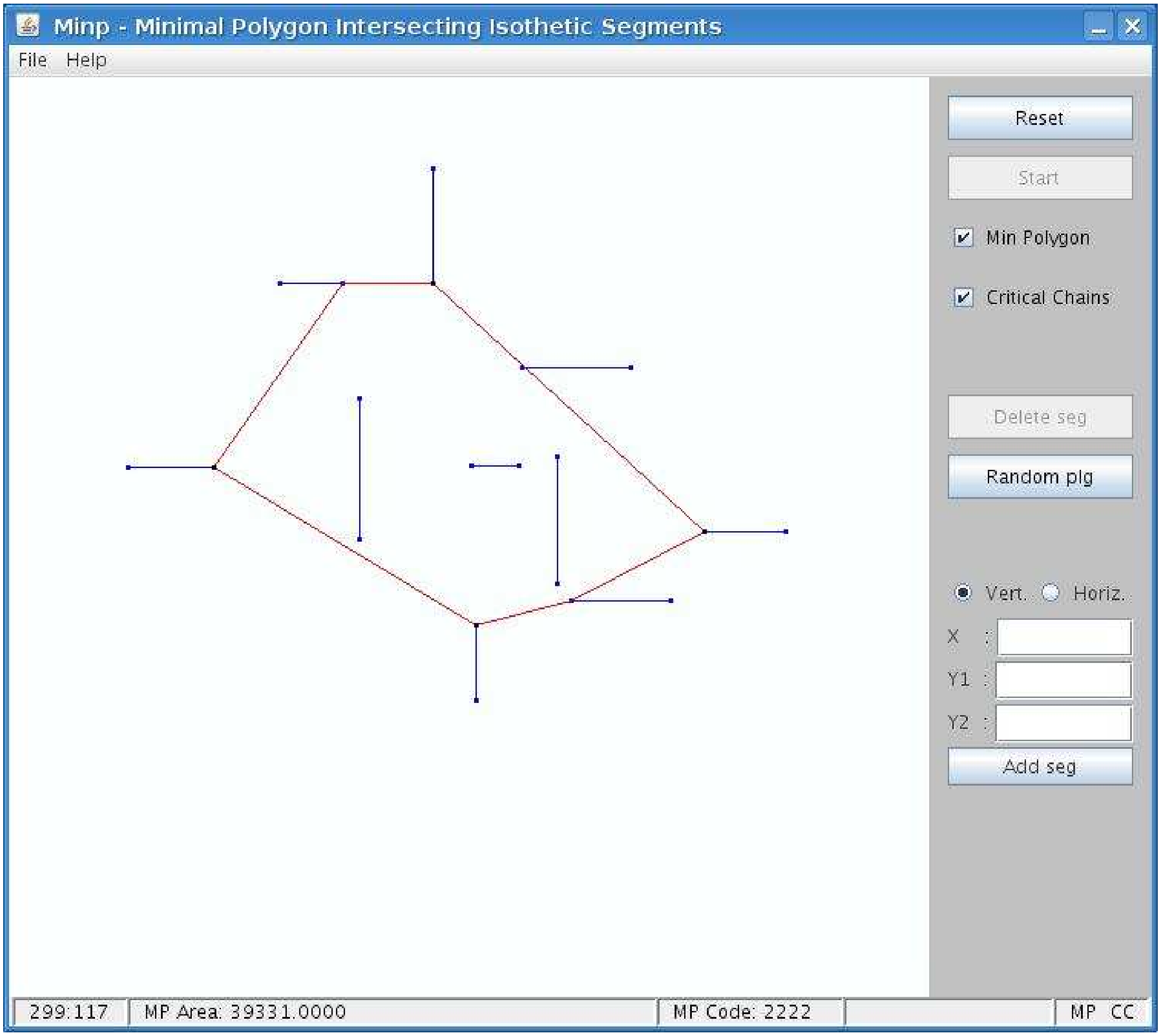


Figure 7: Screen shot of our application

3 Input/output handling

A given set of input segments can be degenerate in a variety of ways. In our implementation, we have tried to handle these situations exhaustively.

Line Stabbing

A line stabber is a line that intersects all the segments in S. When such a line exists, P_{min} reduces to a line segment of zero area. In this case, our implementation simply reports the situation without producing any output.

Our implementation detects the existence of a line stabber based on the characterization of the following Lemma 2. proved in [6].

Lemma 2 *l is a line transversal of the set of isothetic line segments if and only if it is either above the convex hull of the set $S_{RB} = \{\texttt{bot}(s_i), \texttt{right}(s_i) | i = 1, 2, \ldots, n\}$ and below the convex hull of the set $S_{LT} = \{\texttt{top}(s_i), \texttt{left}(s_i) | i = 1, 2, \ldots, n\}$, or below the convex hull of the set $S_{RT} = \{\texttt{top}(s_i), \texttt{right}(s_i) | i = 1, 2, \ldots, n\}$ and above the the convex hull of the set $S_{LB} = \{\texttt{bot}(s_i), \texttt{left}(s_i) | i = 1, 2, \ldots, n\}$.*

Insufficient extreme segments

If the input has fewer than 3 extreme segemnts, the input is not processed.

Multiple extreme segments

While the algorithm assumed that these extreme segments are unique, input data may not satisfy this. Suppose there are multiple horizontal segments that are topmost. We can deal with this situation as follows. Let TH be the set of topmost horizontal segments. Let the left endpoint of each segment be marked RED and the right endpoint of each segment be marked BLUE. Pick the lefmost-BLUE point and the rightmost-RED point. If the rightmost-RED point is to the left of the leftmost-BLUE point the segments in TH have a common intersection that goes from the rightmost-RED to the leftmost-BLUE point and we replace TH by this single segment. Else, our solution for P_{min} will have a top horizontal segment that goes from the leftmost-BLUE point to the rightmost-RED point.

The correctness of this is not hard to prove. If every pair of segments in TH have a common intersection, then by Helly's theorem all the segments in TH have a common inersection. Otherwise, there exists a pair that does not intersect and hence a pair whose left and right end-points are farthest apart. These correspond to the leftmost-BLUE and rightmost-RED points in the above algorithm.

In our implimentation it was very easy to identify the case where the input data contains multiple extreme segments. Handling that situation, however, was not as simple as identifying it. If TH does not contain a common intersection then the RB and LB convex chains intersect one another over a line segment, not a single point. Let TI be the intersection of the two chains, meaning if a point p is covered by RB and is part of the structure of LB, then p is in TI. We then add the points in TI to both chains, which, due to the definition of TI, will not change the shape of either chain. By doing this, we can now safely set either the leftmost-BLUE point, or the rightmost-RED point to be the unique extreme segment, and continue forward with the algorithm.

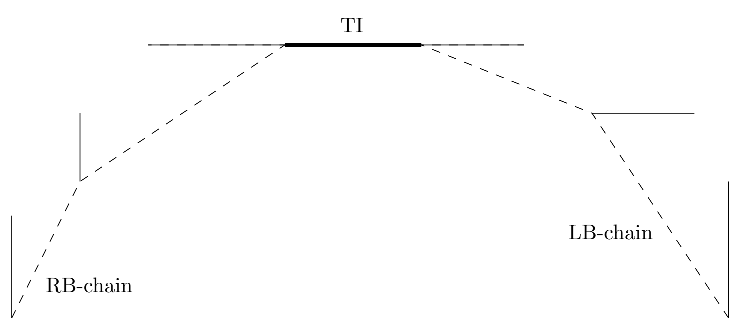


Figure 8: Finding the intersection of two chains

Output handling

The GUI displys P_{min} in red and its area is shown in a text display bar at the bottom. An user can also verify the minimality of the output polygon by sliding the extreme vertices on their respective segments when it is possible to do so. The text display shows the area of the altered polygon that can be checked against the are of P_{min}.

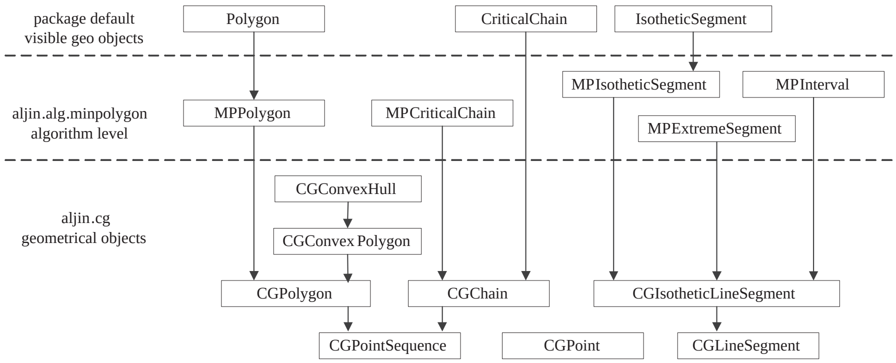


Figure 9: Hierarchy of geometry-related classes

4 Software Architecture

The interconnections of the different packages are shown in the diagram below This software largely uses inheritance, the main feature of OO-programming, to build up the complex functions. The classes designed can be classified into four categories as follows.

1. GUI-purpose (package aljin.guilet): these classes fulfill the GUI interface such as menu bar and dialog windows, deal with interactions between software and user. This package consists of various event listeners and GUI components. For example, class MEasel provides an easy-to-use canvas; while class MArena is a graphic system to visualize user-defined geometrical objects and make them editable.

2. CG-purpose (package aljin.cg): to provide implementations of objects with regard to computational geometry. Typical classes include CGPoint, CGLineSegment, and CGPolygon.

3. Algorithm-purpose (package aljin.alg.minpolygon): implementation of algorithm. This package contains the core functions and some related algorithm-level classes.

4. Application-purpose (package default): these classes compose the executable application. The program entry is Minp.main().

Figure 9 illustrates the hierarchy of geometry-related classes, where solid directed segment means "derived from".

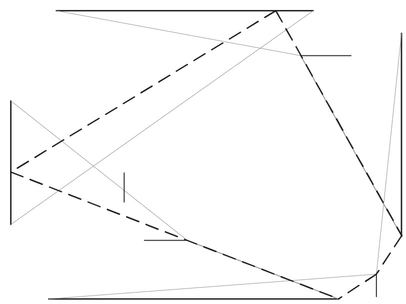
5 Experimental Results

In this section, we provide some experimental results to illustrate the two optimization technologies and the three types of connections.

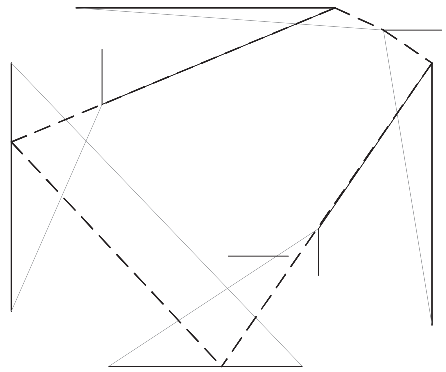
In Figure 10(a), the configuration of the minimal polygon is 0121 and, in Figure 10(b), it is 1012. Since the input of the former example is the outcome of 90 degree counterclockwise rotation of the later one, the application uses the same procedure to compute the minimal polygon with different arguments. In fact, only the order of arguments is different. Similarly, one procedure can be used to compute both 2101 and 1210.

In Figure 10(c), the configration of the minimal polygon is 120 and, in Figure 10(d), the result is 102. Since the two inputs are y-coordinate symmetric, they are handled by the same procedure.

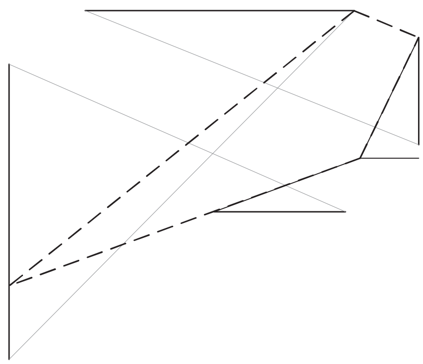
In Figure 10(e), the configuration of the minimal polygon is 12, where the connection with type 1 is a regular one and the connection with type 2 bypasses two extreme segments. In Figure 10, both connections with type 2 bypass one extreme segment.



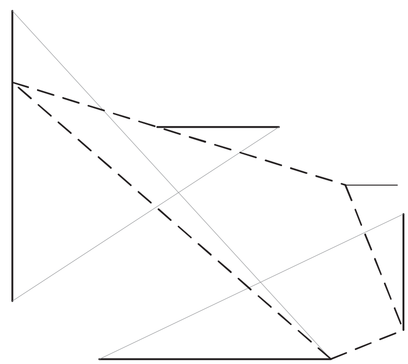
(a) Resulting polygon with code 0121



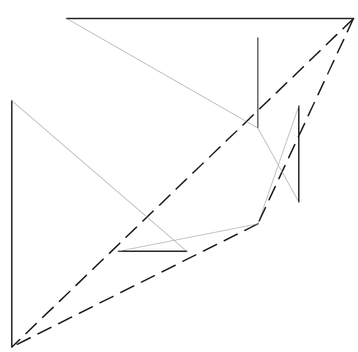
(b) Resulting polygon with code 1012



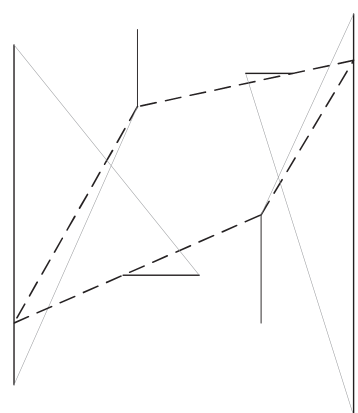
(c) Resulting polygon with code 120



(d) Resulting polygon with code 102



(e) Resulting polygon with code 12



(f) Resulting polygon with code 22

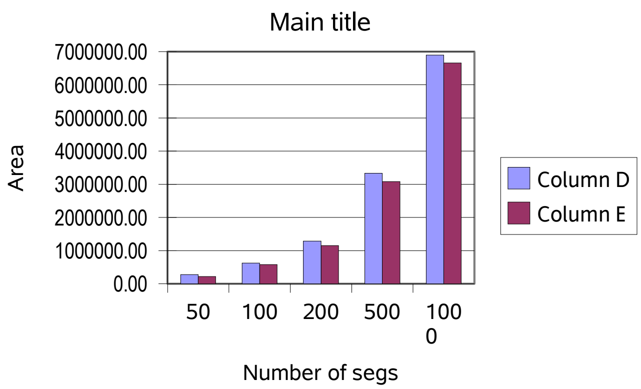
6 Conclusions

The algorithm and its implementation was so complex, that it made us think if it might be worthwhile, in a practical situation, to pick a point at random from each segment and report the convex hull of these n points as an approximation of the minimum area polygon. Repeating and averaging over a large number of runs might work out to be even better. We include a table below from an experiment we made that confirms our point of view. For example, on a 1000 segemnts input, the average area of 50 convex polygon stabbers is within 96,59% of the optimal.

A more sophisticated approach might be to identify a small (constant ?) fraction of the input (a la core sets) that can be used to obatain a good approximate solution. This is worth exploring further.

case#	segs	field*	ave area of 50 RPs	area of MP	exe time(millisec)	MP/RP
1	50	600x600	275351.98	217537.51	719	79.00%
2	100	849x849	624077.98	581302.50	139	93.15%
3	200	1200x1200	1287109.49	1150058.57	194	89.35%
4	500	1897x1897	3334263.90	3081973.66	297	92.43%
5	1000	2683x2683	6895636.97	6660214.78	98	96.59%

* such that the ave space for each seg are the same



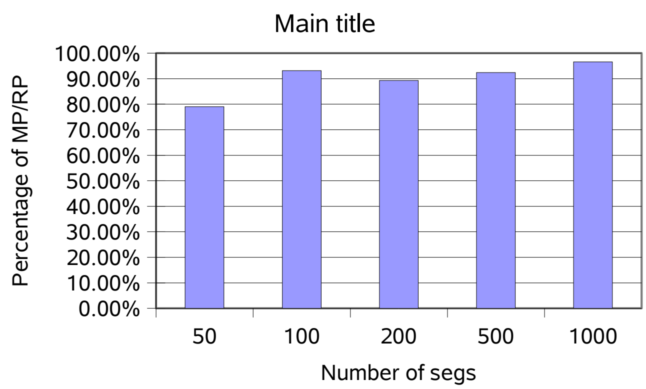


Figure 10: Experiments with a naive algorithm

Acknowledgements

This research was supported by an NSERC Discovery Grant to the third author.

References

[1] O'Rourke. Computational geometry column # 3. *SIGGRAPHB: Computer Graphics (SIGGRAPH)*, 21, 1987.

[2] M. Goodrich and J. Snoeyink. Stabbing parallel segments with a convex polygon. *Computer vision, Graphics and Image Processing*, 49:152–170, 1990.

[3] Asish Mukhopadhyay, Chanchal Kumar, Eugene Greene, and Binay Bhattacharya. On intersecting a set of parallel line segments with a convex polygon of minimum area. *Inf. Process. Lett.*, 105(2):58–64, 2008.

[4] Kelly A. Lyons, Henk Meijer, and David Rappaport. Minimum polygon stabbers of isothetic line segments. Department of Computing and Information Science, Queen's University, Ontario, Canada.

[5] David Rappaport. Minimum polygon transversals of line segments. *International Journal of Computational Geometry & Applications*, 5(3):243–256, 1995.

[6] Asish Mukhopadhyay, Eugene Greene, and S. V. Rao. On intersecting a set of isothetic line segments with a convex polygon of minimum area. To appear in the Intnl. Journal of Comp. Geom. and Applications.

[7] Maarten Loffler and Marc van Kreveld. Largest and smallest convex hulls for imprecise points. Technical Report UU-CS-2006-019, Institute of Information and Computing Sciences, Utrecht University, 2006.

[8] H. Edelsbrunner, H. A. Maurer, F. P. Preparata, A. L. Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. *BIT*, 22:274–281, 1982.