Cutting a Country for Smallest Square Fit

Marc van Kreveld[1] and Bettina Speckmann[2⋆]

[1] Institute for Information and Computing Sciences, Utrecht University,
marc@cs.uu.nl
[2] Institute for Theoretical Computer Science, ETH Zürich,
speckman@inf.ethz.ch

Abstract. We study the problem of cutting a simple polygon with n vertices into two pieces such that – if we reposition one piece disjoint of the other, without rotation – they have the minimum possible bounding square. If we cut with a single horizontal or vertical segment, then we can compute an optimal solution for a convex polygon with n vertices in $O(n)$ time. For simple polygons we give an $O(n^4\alpha(n)\log n)$ time algorithm.

1 Introduction

When browsing through the Rand McNally's Road Atlas of the U.S.A., it appears that not every state is shown on one single or double page. Sometimes the northern and southern halves of a state are shown on two consecutive double pages. Since a state can be modeled by a simple polygon, the aspect ratio of a double page is fixed, and the map scale is supposed to be the same on both double pages, the problem can be modeled as covering a simple polygon by two equal-size axis-aligned squares of smallest size. The squares represent the pages, and by scaling any fixed aspect ratio can be handled. Algorithmically, the problem of covering a simple polygon with two pages is not very difficult and can be solved in linear time. In the case of three map pages, a more involved algorithm can still achieve linear running time, which was shown by Hoffmann [5].

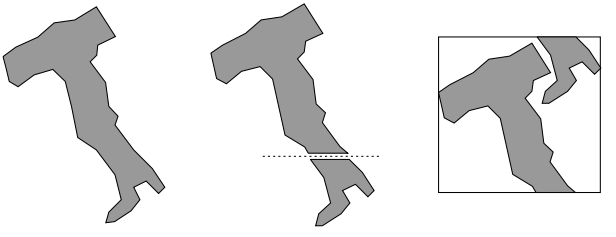

Fig. 1. Fitting a country into a square.

⋆ Supported by the Berlin-Zürich Graduate Program "Combinatorics, Geometry, and Computation", financed by the German Science Foundation (DFG) and ETH Zürich.

P. Bose and P. Morin (Eds.): ISAAC 2002, LNCS 2518, pp. 91–102, 2002.

Another method the Rand McNally's Road Atlas uses to make states fit better on pages is to cut them into two pieces, rearrange these, and fit them on a single or a double page. For example, the western part of Florida can be cut off and placed at a different position to allow a larger map scale. In some edition of the road atlas this technique was applied to seven of the states.

This paper discusses an abstracted version of the problem of cutting a country or state optimally and fitting the pieces on a page. A country is represented by a simple polygon and we cut this polygon with either a horizontal or vertical line segment into two simply-connected pieces. We then rearrange the pieces by translation only and do not allow them to intersect. An optimal cut and an optimal placement are the ones that result in the smallest enclosing square. For a convex polygon with n vertices we compute an optimal solution, i.e., an optimal cut and an optimal placement, in $O(n)$ time. For a simple polygon we give an $O(n^4\alpha(n)\log n)$ time algorithm.

In various papers, Daniels and Milenkovic report results on containment, motivated by marker making in the textile industry. Generally, multiple-part, translational containment problems are NP-hard [2] and known algorithms contain the number of parts in the exponent. Simplified to be comparable to our case, fitting two convex polygons with n vertices inside a constant size polygon takes $O(n)$ time [7], if the polygons are non-convex it requires $O(n^4)$ time [2]. If minimization of the enclosing polygon is of interest, then Milenkovic shows a lower bound of $\Omega(n^4)$ for a related problem [6].

Alt and Hurtado [1] discuss packing convex polygons into a rectangle of smallest size, measured either by area or by perimeter. They study both packing with overlap and without overlap. Of the latter type, they show that a smallest rectangle can be found in linear time for two convex polygons when no rotation is allowed.

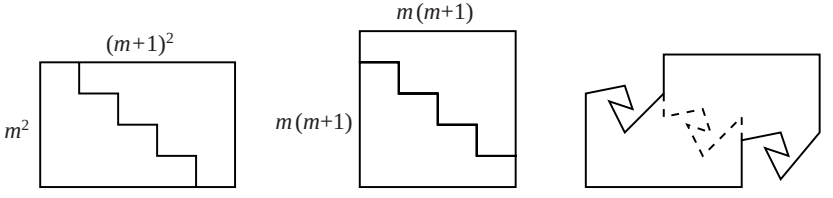

Fig. 2. Dissecting a rectangle and a simple polygon to form a square.

Dissections have been studied at length in Frederickson's book [3]. It contains an interesting example of a rectangle that requires a dissection consisting of many segments so that after repositioning one piece a square is formed, see Figure 2. To the right, there is another example to show that optimal cuts may have a complex shape.

The algorithmic problems and results mentioned above are different from ours because we compute both a cut and a packing into a smallest square. Previous papers compute only a packing (and papers on cutting are less related to ours). Combinatorially, there are a linear number of possible places for a horizontal or vertical cut through a simple polygon. Still our bound in the convex case remains linear and in the non-convex case it is worse only by a factor of $O(\alpha(n) \log n)$ compared with algorithms that do packing only. In our solution to the non-convex case most of the complications do not arise from the linearly many combinatorially distinct cuts to consider, but from discretizing the position of the cut while maintaining the optimality and efficiency.

In Section 2 we present the algorithm for convex polygons and in Section 3 we describe the algorithm for non-convex polygons. We conclude with some open problems.

2 Cutting a Convex Polygon

Assume we are given a convex polygon P with n vertices. Our goal is to find a vertical line segment C that cuts P into two parts A and B and then to translate A to a position in which A and B do not properly intersect, such that the smallest enclosing axis-parallel square S for A and B is minimized. If we also allow horizontal cuts, then we repeat the algorithm with x- and y-coordinates exchanged.

In order to efficiently find an optimal solution, i.e. a configuration that is determined by an optimal cut and an optimal placement, we identify certain properties of a subset of the optimal solutions. In particular, we show that there is always an optimal solution such that (i) either A or B determine the x or the y-span of S, (ii) A and B are in contact, and (iii) both A and B are in contact with either the left or the right side of S (see Lemma 1). Restricting ourselves to search for optimal solutions that have the properties just described and incorporating the following two observations yields a surprisingly simple algorithm that computes an optimal solution in linear time.

First, we define the NW chain of a convex polygon Q to be the polygonal chain that is the part of the boundary of Q from the leftmost vertex to the topmost vertex, clockwise. The SW chain, NE chain, and SE chain are defined similarly. Horizontal and vertical edges are included in the chain their leftmost endpoint – clockwise – is adjacent to.

Observation 1 *Given two convex polygons Q and R, and a square S, such that $s = \max\{x\text{-span}(R), y\text{-span}(R)\}$, then if Q and R both fit in S, there is a placement where Q (i) touches the left side of S, and R in the NW chain, or (ii) touches the left side of S, and R in the SW chain, or (iii) touches the right side of S, and R in the NE chain, or (iv) touches the right side of S, and R in the SE chain.*

Applied to A and B this observation translates to: If B is the polygon that determines the span of S then there are only four canonical placements for A.

Second, we can observe that if we consider an optimal solution which has the properties described above, then while shifting the cut to shrink one polygon and grow the other, always keeping them in contact, the contact moves only in one direction along the boundary of each polygon, passing over each vertex at most once. For a specific constellation this observation translates to:

Observation 2 *If A and B fit in S, and the x-span of B is s, and A is in contact with the left side of S and the NW chain of B, then when the cut is shifted to shrink B and grow A, the contact between A and B can only go rightward on B (clockwise) and rightward on A (counterclockwise), assuming A keeps touching the left side of S and the NW chain of B.*

Similar observations hold if A is in contact with any other chain of B or if B determines the y-span of S.

Our algorithm now actually consists of eight incremental algorithms: The cut starts four times at the left side of P, implying that B is the larger polygon that determines the size of S. The other four times the cut starts at the right side of P and A is the larger polygon. If we start at the left side we initialize with $A = \emptyset$ and $B = P$ and grow A and shrink B (therefore also S) while keeping A to the NW (resp. SW, NE, or SE) of B. We then start with $A = P$ and $B = \emptyset$ and grow B and shrink A (therefore also S) while keeping B to the NW (resp. SW, NE, or SE) of A. During the run of each algorithm we maintain the contact points on A and B and a bounding square S whose size corresponds to the x or y-span of either A or B depending on the constellation we are currently processing. The algorithms terminate whenever A and B do not fit into S anymore.

There are two types of events our algorithm has to process: (i) the cutting line passes over a vertex of A or B and (ii) the contact point between A and B proceeds to a next vertex or edge on A or B. Based on the observations above it is straightforward to see that each algorithm only needs to process $O(n)$ events, each at constant cost.

Whenever an algorithm terminates we use the current information on the contacts between A, B, and S to compute the optimum in-between the last two events. Finally, the minimum of the minimal square sizes found by the eight algorithms is the true minimum square size we set out to find.

Now all that remains is to prove the following lemma:

Lemma 1. *For a given convex polygon P there exists an optimal vertical cut C and an optimal translation of A and B that put A and B without intersection into a square S with side length s and one of the following holds:* **(a)** *the x-span or y-span of B is equal to s and if A is not empty, then it is in contact with B and with the left or right side of S or* **(b)** *the x-span or y-span of A is equal to s and if B is not empty, then it is in contact with A and with the left or right side of S.*

Proof. (sketch) Assume that an optimal solution is given, where S is the enclosing square with side length s, the pieces are polygons A and B, and the cut is C. We will transform this solution into one that satisfies the statement in the lemma, without increasing the side length s of S.

Assume first that the angles of the cut edge corners α_t and α_b of A sum up to at most π. Then we will show how to transform to case **(a)** of the lemma. Otherwise, the angles of the cut edge corners β_t and β_b of B sum up to at most π because P is convex. Then it follows by symmetry that we can transform to case **(b)**. Hence we only need to show the first part, and we assume that $\alpha_t + \alpha_b \leq \pi$.

Let ℓ be a line that separates the interiors of A and B. First assume that ℓ can be vertical. We shift the cut edge to make B grow and A shrink simultaneously. This cannot increase the x-span of A and B, so we can continue until either A becomes empty and $B = P$, or the y-span of B becomes s. In both cases we are done.

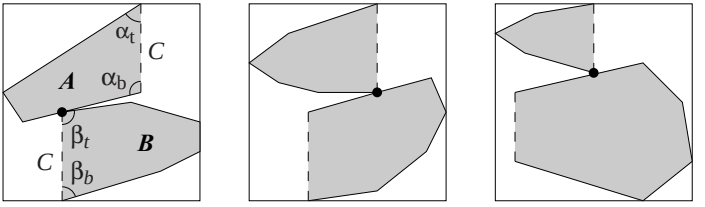

Fig. 3. The three cases of the proof.

Now assume that ℓ cannot be vertical. By symmetry we may assume that A lies above ℓ and B lies below ℓ, see Figure 3. Again we need to consider two cases, one where ℓ has positive slope and one where ℓ has negative slope. These cases are not symmetric. However, in this sketch we will discuss only one. Furthermore, the case of a horizontal line ℓ follows in exactly the same way, but here it is not treated explicitly because it increases the complexity of the formulations.

Since B is below and right of ℓ, and A above and left of ℓ, we can assume that S and B are in contact at both the bottom and right side of S, otherwise we can move B to make this true. Similarly, we can assume that A is in contact with the top and left side of S, or we can move A inside S to make this happen. If A and B can be separated by a vertical line in this new configuration, then we are done as just shown, so we assume that this is not the case. If A and B are in contact, then the tangent point can only be the lower endpoint of the cut edge of A or the upper endpoint of the cut edge of B (see the left two pictures in Fig. 3), otherwise we obtain a contradiction with the convexity of P. Furthermore, because of the positive slope of ℓ, the angle assumption on A, and the fact that A and B come from one convex polygon, we can show that the highest point of A is the upper endpoint of the cut edge.

The main observation is that we can shift the cut to grow B and shrink A simultaneously, until the either the x-span or the y-span of B is equal to s. During this shift, we will not move A, B, or S in horizontal direction, only vertically. We must show that A and B still fit in S vertically, until the x-span

or y-span of B is s. Then we can reposition A to satisfy the other criteria of the lemma.

We next show that the cut can be shifted without forcing the square S to increase in size. When shifting the cut leftward, A loses a trapezoidal region bounded from the left and right by vertical sides. Because $\alpha_b + \alpha_t \leq \pi$, the left side of the trapezoid is at most as long as the right side. Because of the shift, piece A can move upward in S with an amount depending on the slope of the edge of A counterclockwise from the cut edge, and the amount of shifting. This follows from the observation that the highest point of A must be the upper endpoint of the cut edge of A. At the same time, B grows, and pushes itself and/or A upward with respect to the bottom side of S. If A and B were not in contact yet, they may come in contact. Three possible situations are shown in Figure 3. Which contacts occur, the slopes of the edges clockwise and counterclockwise of the cut edge in A (not B!), together with the amount of shifting, determine the amount with which A and/or B must move upward due to the size increase of B. Since $\alpha_b + \alpha_t \leq \pi$, part A can move up at least as much as B need be moved up to stay inside S. The sum $\alpha_b + \alpha_t$ can only decrease when the cut shifts leftward, so we can continue until the x-span or y-span of B is s.

The arguments of the proof when the separating line ℓ has negative slope are similar and we omit them here. After arriving in the situation where the x-span or y-span of B is s, either A is empty, or we can move A until it is in contact with B. □

Theorem 1. *Given a convex polygon P with n vertices, we can determine a vertical cut C of P into subpolygons A and B, and a non-intersecting placement of A with respect to B, such that the smallest enclosing square of A and B is minimized in $O(n)$ time.*

3 Cutting a Simple Polygon

Assume we are given a not necessarily convex simple polygon P with n vertices. Our goal is to find a vertical line segment C that cuts P into two parts A and B and then to translate A to a position in which A and B do not properly intersect, such that the smallest enclosing (axis-parallel) square S for A and B is minimized.

In an optimal solution A and B will be in contact (more precisely, there is an optimal solution with A and B in contact). Furthermore, we can assume that a vertex v of A lies on an edge e of B. We first choose v and e in P and then consider all cuts C that partition P such that v and e are part of different subpolygons. Consider a vertical decomposition of P: There are a number of *separating trapezoids* such that a cut through any of these trapezoids separates v and e. The separating trapezoids can be linearly ordered such that a cut through the first one results in the smallest A and largest B and cuts through the following trapezoids grow A while shrinking B. In Figure 4 the separating trapezoids are shaded grey and ordered from left to right. For convenience we

add the vertices of the trapezoidal decomposition as vertices to (the boundary of) P. This does not restrict the problem and the number of vertices of P is still in $O(n)$. The edge e is now an edge of the polygon including the vertices induced by the trapezoidation, so it can be a subedge of an edge of the original input polygon (see Fig. 4).

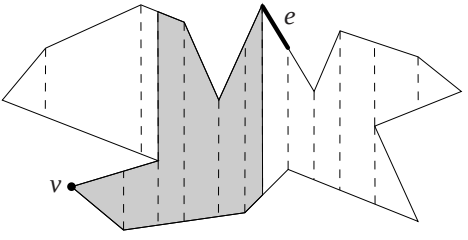

Fig. 4. A polygon P with v and e chosen; the separating trapezoids are shaded grey.

We first describe only the cases where the edge e of B lies on the boundary of P and is not the one cut edge of B. Similarly, we assume first that v is not an endpoint of the cut edge of A and that e is not adjacent to the cut edge of B. We make these assumptions to explain the general idea and we will show later how to handle all cases.

All positions with $v \in A$ in contact with $e \in B$ can be represented by an interval I of the real line. At each end of I, vertex v coincides with one of the endpoints of e. For ease of description, we assume B to be stationary and A translating with respect to B. Since v must be in contact with e, A will translate along the vector between the two endpoints of the edge e. We are interested in those positions where A and B are non-intersecting. A and B intersect if and only if they each have a line segment that properly intersect. In a degenerate case this may not be true, but we will not consider such cases here. If we consider one line segment e_A of A in translation, then in the general case, e_A intersects a (steady) line segment e_B of B along some stretch, but possibly not before and/or not after it. The positions on interval I where e_A and e_B intersect are a subinterval of I. Since this holds for all edges of A and B, there are $O(n^2)$ intervals on I that define in what position of A with respect to B they intersect. We will store these intervals in a segment tree, and augment every internal node μ with a boolean FREE that indicates whether in the interval I_μ of I, represented by μ, there is at least one position that is not covered by any of the $O(n^2)$ intervals in the subtree rooted at μ. Since this boolean is only valid for the intervals stored in the subtree rooted at μ, there may be an interval stored at an ancestor of μ which makes no position for μ non-covered. The boolean helps to answer queries with a query interval I_q, to locate the leftmost or rightmost non-covered position in I_q, in $O(\log n)$ time. We will use the term "free" throughout the description for a placement of A with respect to B so that they do not properly intersect.

We do not have to store the intervals explicitly at the nodes of the segment tree. Instead, we store a counter at each node that represents how many intervals are stored at that node as in a segment tree for stabbing counting queries. The counts and augmentation by booleans can be maintained under insertion and deletion of any interval in $O(\log n)$ time. The segment tree needs $O(n^2)$ storage because it represents $O(n^2)$ intervals.

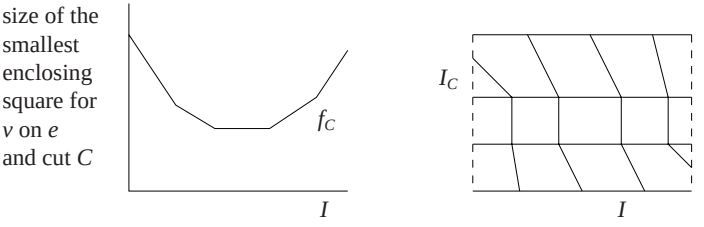

Fig. 5. The function f_C for a given cut C (left); the patches for all positions on I and cuts inside a trapezoid (right).

Assuming that the position of the cut C is fixed, let f_C be the function that maps the position of v on e (or, equivalently, a point on I) to the side length of the smallest enclosing square of A and B in the specified position. The function f_C is defined on every point of I, regardless whether this position is free. Over the interval I f_C is a piecewise linear continuous function consisting of at most five pieces (see Fig. 5). If there are five pieces, then the first two pieces have negative slope, the middle piece is horizontal and is the minimum of the function, and the last two pieces have positive slope. The function f_C is easy to compute in $O(n)$ time. The optimal free placement for A and B with v on e is either some position on I that realizes the minimum of f_C, or the rightmost position on I left of the minimum, or the leftmost position right of the minimum. Note that with the help of the segment tree we can determine the optimum placement inside any query interval I_q in $O(\log n)$ time with at most three queries. We query with the subinterval of I_q that has negative slope to find the rightmost free position, we query with the subinterval of I_q with constant slope for any free position, and we query with the subinterval of I_q with positive slope for the leftmost free position.

Assume next that the cut C jumps from trapezoid boundary to trapezoid boundary. Assume first that this makes A larger by exactly one trapezoid and B smaller by that same trapezoid. More generally, it can happen that a large part of B suddenly goes to A, like the upper left part of P in Figure 4. We can treat this as a sequence of single trapezoids going from B to A. This may make A temporarily disconnected, but this will not affect the algorithm. We will not query the segment tree for a free position until we reach the next possible cut. When we jump to a next trapezoid boundary A loses its former cut edge, it gains

a new cut edge, and it gains two edges from B. Similarly, B loses its former cut edge and two more edges and gains a new cut edge. In total, four edges change for each of A and B. These eight edges were involved in $O(n)$ of the $O(n^2)$ intervals on I. To update the segment tree, we perform $O(n)$ insertions and deletions of intervals. Since the FREE-information and the counts in the segment tree can be maintained in $O(1)$ time per node visited, one update takes only $O(\log n)$ time. Then we have the segment tree for the next cut. We can determine the new function f_C and query for the new optimum with three queries. In total, taking one whole trapezoid from B and adding it onto A takes $O(n \log n)$ time.

In general it is the case that the optimum cut and position of v on e does not occur at a trapezoid boundary, but somewhere in the middle between two boundaries. We handle this as follows: We consider simultaneously the position of v on e and the exact position of the cut between two consecutive trapezoid boundaries. When the cut progresses from one trapezoid boundary to the next, we could maintain all changes that occur to the $O(n^2)$ intervals on I. However, there can be a cubic number of events (endpoint swaps) between intervals that change and intervals that do not change – this is too costly to maintain. Instead we keep changing and non-changing intervals separate in our solution. We represent the exact position of the cut as a second dimension added to I, which yields another interval I_C, and we consider the rectangular region $R = I \times I_C$ (see Fig. 5). Any point in this region represents a position of v on e and a position of the cut in the trapezoid under consideration. The lower side of R corresponds to the situation where the cut trapezoid is completely part of B and the upper side corresponds to the situation where the cut trapezoid is completely part of A. The $O(n^2)$ intervals resulting from one edge of A and one edge of B now become regions of one of two types, see Figure 6. These regions show where

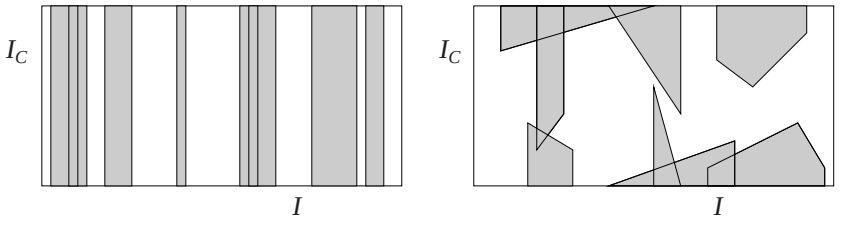

Fig. 6. The rectangle $R = I \times I_C$: the rectangular forbidden regions (left) and all other forbidden regions (right).

the edges intersect, so they are forbidden regions for the free placement of A with respect to B. If edges $e_A \in A$ and $e_B \in B$ do not change when the cut is moved in the trapezoid, then we get a rectangular region $i \times I_C$ in R, where i was the original interval we got for e_A and e_B. There are $O(n^2)$ such rectangles. All pairs of edges that involve the cut or one of the edges that go from B to

A when the cut progresses, define $O(n)$ differently shaped regions. The regions involving an edge adjacent to the cut edge of A are triangular or quadrilateral, and more specifically, are the region vertically above some line segment inside R. This line segment may have one endpoint on the upper side of R, which determines whether it is triangular or quadrilateral of shape. Similarly, any region involving an edge adjacent to the cut edge of B is the region vertically below a line segment.

The regions involving the cut edge of A or B are also simple polygonal shapes. However, their shape is such that a vertical line could intersect them twice inside R and therefore the complexity of the boundary of the forbidden regions inside R may be quadratic, even if we do not consider the rectangles. Due to the complexity of these regions the solution we are about to describe would not be efficient, so we need to represent them differently. We make use of the following simple geometric observation (see Fig. 7):

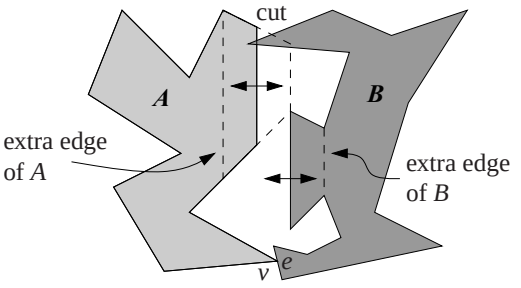

Fig. 7. Vertex of B inside the cut trapezoid of A.

Observation 3 *If polygons A and B intersect and the only edge of A involved in the insection is the cut edge of A, then there must be a vertex of B inside A. Furthermore, if the left extreme position of the cut edge is considered a fixed edge of A and this edge also does not intersect any edge of B, then the cut trapezoid of A must contain a vertex of B.*

This observation shows that the only other forbidden regions we must add are the ones where the cut trapezoid of A contains a vertex of B or the cut trapezoid of B contains a vertex of A. This is true if we consider the left side of the cut trapezoid of A to be a fixed edge of A and the right edge of the cut trapezoid to be a fixed edge of B. The extra edge of A was actually used when we considered the situation at the trapezoid boundary just before. So the corresponding intervals are already in the segment tree. For the extra edge of B we insert $O(n)$ intervals in the segment tree.

The new forbidden regions that spring from this observation are the pentagonal regions vertically above or below two adjacent line segments. Summarizing, we have three types of forbidden regions inside R:

1. $O(n^2)$ vertical rectangles inside R, that cross R completely in the vertical direction. These are represented in the segment tree.
2. $O(n)$ regions vertically above a line segment inside R.
3. $O(n)$ regions vertically below a line segment inside R.

We compute the lower envelope of the regions of the second type and the upper envelope of the regions of the third type. Only the regions in-between can contain placements for A and B and a choice of the cut, such that A and B do not intersect. We combine the envelopes by a left-to-right scan to compute the regions in-between. These regions are bounded by at most $O(n\alpha(n))$ line segments and can be computed in $O(n \log n)$ time [4].

The function f that gives the side length of the smallest enclosing square as a function of a point in R (a position of v on e and position of the cut) is now a piecewise linear bivariate function (see Fig. 5). If the cut is fixed, then the function f_C has the shape we noted before. This holds for every cut. The function f may have up to fifteen patches. Two horizontal lines partition the rectangle R into three slabs, the middle slab is partitioned by four vertical lines, and the other two slabs are partitioned by four diagonal lines. The twelve lines partitioning the three slabs connect to each other. The function f gives the side length of the enclosing square regardless of whether A and B intersect.

We now overlay the $O(1)$ patch boundaries of f with the $O(n\alpha(n))$ line segments of the region in-between the envelopes. We take the collection S of all $O(n\alpha(n))$ edges we have in this arrangement, take their x-extents, and query in the segment tree to find both the leftmost and the rightmost free position in the query interval. We use the function f to determine the size of the smallest enclosing square for each answer in $O(1)$ time. In total, the queries to find candidate solutions take $O(n\alpha(n) \log n)$ time. We select the smallest answer, which is the best solution for a given vertex v in contact with a given edge e and a given cut trapezoid.

Lemma 2. *The optimal placement and cut for a given v, e, and cut trapezoid is a leftmost or rightmost free placement on a segment of S.*

Proof. By construction, any segment of S lies inside of only one patch of f or is part of a patch boundary. Over each patch a linear function defines the side length of the smallest enclosing square. Obviously, we need a free placement so that A and B do not intersect. The lemma follows. □

We already described how to update the segment tree so that it becomes valid for the next separating trapezoid. The update requires $O(n \log n)$ time in total and since we go through at most $O(n)$ trapezoids, the total update time becomes $O(n^2 \log n)$. The initial costs for constructing the segment tree are also $O(n^2 \log n)$, which brings the total cost for one edge e, one vertex v, and all cuts that separate them to $O(n^2\alpha(n) \log n)$. Concluding, if the optimal cut of

polygon P and position of the parts A and B has some vertex-edge contact not at the cut edge, then we can determine it in $O(n^4\alpha(n)\log n)$ time.

The remaining issue is a situation where the optimal cut and position has a contact between A and B that involves an edge or vertex of the cut edge of A or B. A careful case analysis shows that this situation can be handled without affecting the running time of our algorithm asymptotically. Due to the limited amount of space available for this abstract we omit the details of this analysis. However, they can be found in the full version of the paper.

Theorem 2. *Given a simple polygon P with n vertices, we can determine a vertical cut C of P into subpolygons A and B, and a non-intersecting placement of A with respect to B, such that the smallest enclosing square of A and B is minimized in $O(n^4\alpha(n)\log n)$ time and $O(n^2)$ space.*

4 Open Problems

The problem of cutting a polygon into two and rearranging the parts gives rise to various interesting and very difficult problems. Our algorithm breaks down if we allow straight line cuts that need not to be horizontal or vertical. An even more general problem arises if we allow several line segments or even any curve as the cut and still wish to rearrange the parts optimally to fit inside a square. It might also be of interest to study the problem if we do not restrict ourselves to translations but also allow rotations while rearranging the parts.

References

1. H. Alt and F. Hurtado. Packing convex polygons into rectangular boxes. In *Discrete and Computational Geometry – Japanese Conference, JCDCG 2000*, number 2098 in Lect. Notes in Comp. Science, pages 67–80. Springer, 2001.
2. K. Daniels and V. J. Milenkovic. Multiple translational containment, part i: An approximate algorithm. *Algorithmica*, 19(1–2):148–182, September 1997.
3. Greg Frederickson. *Dissections: Plane and Fancy.* Cambridge University Press, 1997.
4. J. Hershberger. Finding the upper envelope of n line segments in $O(n\log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.
5. Michael Hoffmann. Covering polygons with few rectangles. In *Abstracts 17th European Workshop Comput. Geom.*, pages 39–42. Freie Universität Berlin, 2001.
6. V. Milenkovic. Translational polygon containment and minimal enclosure using linear programming based restriction. In *Proc. 28th Annu. ACM Sympos. Theory Comput.*, pages 109–118, 1996.
7. Victor J. Milenkovic. Multiple translational containment, part ii: Exact algorithm. *Algorithmica*, 19(1–2):183–218, September 1997.