Information Processing Letters 33 (1989/90) 35–44
North-Holland
27 October 1989

AN APPROXIMATE ALGORITHM FOR THE MINIMAL VERTEX NESTED POLYGON PROBLEM

Bhaskar DASGUPTA *
CMC Ltd., R & D, Secunderabad 500 003, India

C.E. VENI MADHAVAN
Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India

Communicated by R. Wilhelm
Received 11 July 1989
Revised 12 June 1989

Given two simple polygons, the Minimal Vertex Nested Polygon Problem is one of finding a polygon nested between the given polygons having the minimum number of vertices. In this paper, we suggest efficient approximate algorithms for interesting special cases of the above using the shortest-path finding graph algorithms.

Keywords: Computational geometry, graphs, shortest paths

1. Introduction

Given two simple polygons P and Q having p and q sides respectively, with Q inside P, the Minimal Vertex Nested Polygon Problem (MVNP problem, for short) is the one of finding a polygon K of k sides nested between P and Q such that, for any other polygon L of l sides nested between P and Q, $k \leqslant l$. When both P and Q are convex, Aggarwal et al. [1] give an $O((p+q)$ time approximate algorithm and an $O((p+q)\log k)$ time exact algorithm. For the general case when both P and Q are arbitrary simple polygons, Suri and O'Rourke [9] give an $O(p+q)^2)$ time exact algorithm. We consider the following special cases:

(A) When both P and Q are rectilinearly convex (not necessarily rectilinear) polygons, we suggest an $O(n \log n + e)$ time approximate solution for the problem, where $n = p + q$ and e is the number of edges of the visibility graph associated with this problem, $e = O(n + n^2 - nk)$.

(B) When P is rectilinearly convex and Q is an arbitrary simple polygon, we suggest an $O(n + m \log m + e)$ time approximate solution for the problem, where $n = p + q$, $m = p + q'$, q' the number of vertices in the rectilinear convex hull of Q, and e is the number of edges of the visibility graph associated with this problem, $e = O(m + m^2 - mk)$.

In both cases above the approximate polygon may have at most $2k$ vertices.

We also show that for the special case when Q is a linearly separable x-convex (or y-convex) polygon, q' is constant on the average, thereby greatly reducing the average size of the visibility graph for special cases of case (B) above. In fact, the result $q' = O(1)$ on the average is proved for a separable monotone polygon, i.e. the direction of monotonicity need not be horizontal or vertical only.

* Present affiliation: Computer Science Department, The Pennsylvania State University, University Park, PA 16802, USA.

The MVNP problem has interesting applications in motion planning in robotics and in VLSI designs. For example, the problem of moving a point object or a circle through a channel with a minimum number of bends is nothing but the MVNP problem. In [7] Maddila discusses algorithms for moving an object through an L-shaped corridor infinite in both directions. The rectilinearly convex case of the MVNP problem solves the case of moving a point or circular object through a channel composed of successive L-shaped corridors.

2. Preliminaries

A simple polygon P is a sequence of vertices and edges such that the edges form a cycle and no two nonconsecutive edges intersect. We denote the interior and boundary of P by $\text{int}(P)$ and $\text{bd}(P)$ respectively. We use the term polygon to mean a simple polygon only.

A polygon is *rectilinear* if its sides are horizontal or vertical only. A polygon P is *x-convex* (*y-convex*) if the intersection of any horizontal (resp. vertical) segment with $\text{int}(P)$ is a connected (possibly empty) segment. A polygon which is both x-convex and y-convex is a *rectilinearly convex* (RC, for short) polygon. An RC polygon need not be rectilinear. An RC polygon which is also rectilinear is a rectilinear and rectilinearly convex (RRC, for short) polygon. The *rectilinear convex hull* (RC hull, for short) of a polygon P, denoted by $\text{RCH}(P)$, is the minimal RC polygon containing P.

Given a polygon P, a *chain* from point s on $\text{bd}(P)$ to point t on $\text{bd}(P)$, denoted by $\text{ch}(s, t)$, is a subset of $\text{bd}(P)$ composed of all edges of P from s to t in counterclockwise order. Hence,

$$\text{bd}(P) = \text{ch}(s, t) \cup \text{ch}(t, s)$$

for all $s, t \in \text{bd}(P)$, $s \neq t$. A point t on $\text{bd}(P)$ is said to be *visible* from a point s on $\text{bd}(P)$ if the line segment from s to t does not intersect $\text{ch}(s, t)$ or $\text{ch}(t, s)$. A chain C is *monotone* with respect to a line l if a line orthogonal to l intersects C at at most one point. A simple polygon is monotone if its boundary can be decomposed into two chains C_1 and C_2 monotone with respect to the same line. A monotone polygon is *linearly separable* if its boundary can be decomposed into two chains C_1 and C_2 monotone with respect to the same line l such that C_1 and C_2 lie on opposite sides of l (Fig. 1).

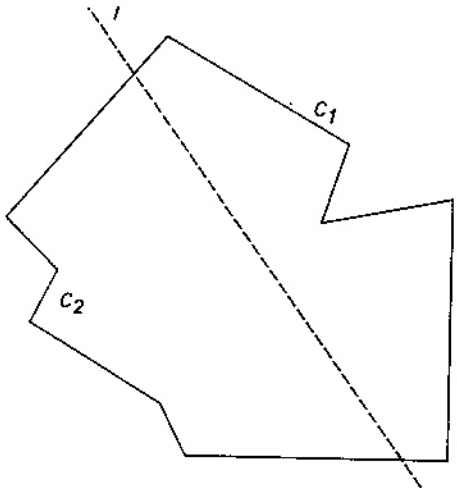

Fig. 1. Linearly separable monotone polygon.

3. Basic algorithm

First, we consider a very special case of the MVNP problem when both P and Q are RRC polygons. We show later how to extend this algorithm for the other cases of MVNP problem, i.e. when P and Q are both RC or Q is arbitrary simple and P is RC. We describe below the basic algorithm.

Algorithm 3.1

Input: Two RRC polygons P and Q, with Q inside P.

Output: The approximation to the actual minimal nested polygon between P and Q.

Algorithm:

Step 1: Divide the region $\text{int}(P) - \text{int}(Q)$ into a number of rectangular slabs $S_1, S_2, \ldots, S_t$ by extending the horizontal edges of P and Q (Fig. 2). Here, S_1 is the topmost or bottommost slab and slabs are labelled starting from S_1 in clock-

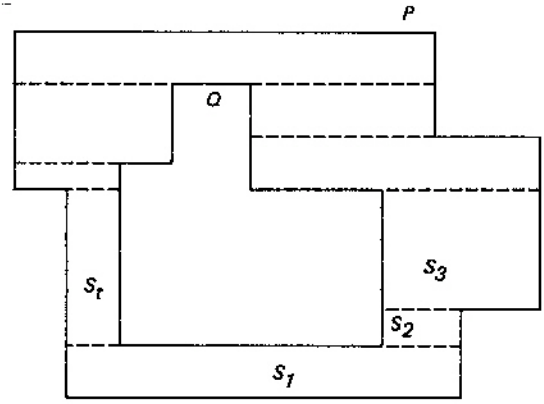

Fig. 2. Division of int(P) − int(Q) into slabs $S_1, S_2, \ldots, S_t$.

wise or counterclockwise order. The bottom (resp. top) of a slab S_i, denoted by B_i (resp. T_i), is that portion of its bottommost (resp. topmost) horizontal edge which does not belong to bd(P) or bd(Q) (Fig. 3).

Step 2: Construct a directed cyclic graph $G = (V, E)$ showing the visibilities of slabs. Hence, $V = \{S_i \mid 1 \leqslant i \leqslant t\}$ and $E = \{(S_i, S_j) \mid j > i, S_j$ is visible from $S_i\}$. Slab S_j is visible from slab S_i if we can draw a straight line from some point of T_i to some point of T_j which does not intersect any of the sides of S_{i+1} to S_{j-1} belonging to bd(P) or bd(Q); this straight line is called a *visibility line* from S_i to S_j. Note that adjacent slabs are always visible and hence there is always one directed cycle in G.

Step 3: Find the shortest directed cycle in G from S_1 to S_1 where S_1 is the bottommost (or topmost) slab. Construct the polygon corresponding to this path. This gives an approximate solution.

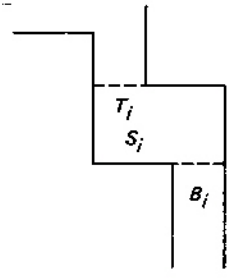

Fig. 3. B_i and T_i shown dotted.

4. Analysis of Algorithm 3.1

We omit the simple proofs of the following two propositions.

Proposition 4.1. *When both P and Q are RC polygons, the minimal polygon K is also RC.*

Proposition 4.2. *The minimal polygon K is RC even if only the outer polygon P is RC.*

Lemma 4.3. *Let k' be the number of edges in the shortest cycle of G in Algorithm* 3.1 *from S_1 to S_1. Let k be the number of vertices in the actual minimal vertex nested polygon K. Then $k' \leqslant k \leqslant 2k'$.*

Proof. This follows from the following observations.

(1) Consider any three consecutive slabs S_p, S_q, S_r, $r > q > p$, in the shortest cycle found by Algorithm 3.1. So, S_q is visible from S_p, and S_r from S_q. If at least one visibility line from S_q to S_r intersects at least one visibility line from S_p to S_q inside int(P) − int(Q), then we can go from S_p to S_r in one bend (Fig. 4); otherwise, we take two bends (Fig. 5).

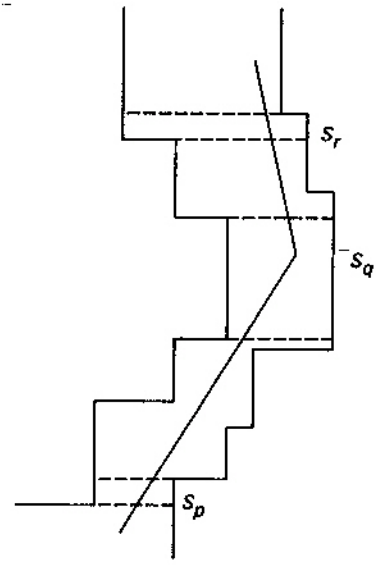

Fig. 4.

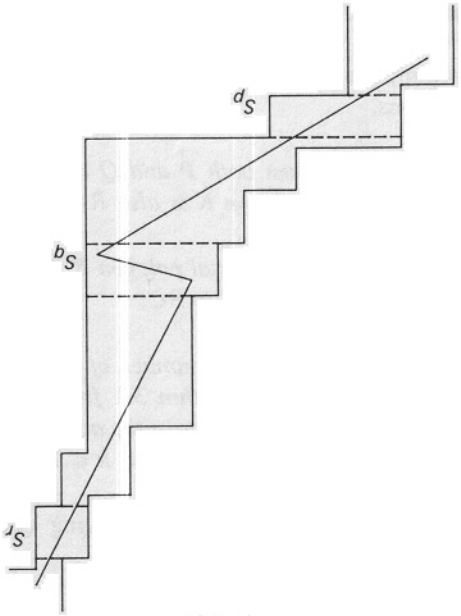

Fig. 5.

(2) In view of observation (1) above, $k \leqslant 2k'$.

(3) $k' \leqslant k$ since a minimal nested polygon defines a cycle of length at most k. □

Lemma 4.4. *The approximate polygon as generated by Algorithm* 3.1 *may have at most* $2k$ *vertices.*

Proof. Since $k' \leqslant k$ and a cycle of length k' can produce a polygon of at most $2k'$ vertices (observation (1) in Lemma 4.3), the approximate polygon may have at most $2k$ vertices. □

Lemma 4.5. *The graph* $G = (V, E)$ *as constructed in Algorithm* 3.1 *may have* $O(n + n^2 - nk)$ *edges, where* $n = p + q$ *and* k *is the number of vertices of the actual minimal polygon* K.

Proof. Let the slabs in Algorithm 3.1 be $S_1, S_2, \ldots, S_t$. Obviously, $t \leqslant \frac{1}{2}(p + q)$. We define the visibility length VL_i of a slab S_i as $|\{S_j \mid S_j$ is visible from S_i, $j > i\}|$. Obviously, $1 \leqslant VL_i \leqslant t - 1$ for $1 \leqslant i \leqslant t$. Now, since adjacent slabs are always visible and K is the actual minimal polygon, VL_i must satisfy the inequality $2(t - VL_i) + 1 \geqslant k$, since we can move from S_i to S_{i+VL_i} directly through one edge and then traverse adjacent slabs with a maximum of 2 vertices at each of the remaining $2(t - VL_i)$ slabs by observation (1) of Lemma 4.3. Hence,

$$|E| = \sum_{i=1}^{t} VL_i = O(n + n^2 - nk). \quad \square$$

Lemma 4.6. *Algorithm* 3.1. *takes* $O(n \log n + e)$ *time where* $n = p + q$ *and* $e = O(n + n^2 - nk)$.

Proof. Step 1 takes $O(n)$ time. We show in Section 5 that Step 2 takes $O(e)$ time. Step 3 takes $O(|V| \log |V| + |E|)$ time [2] and since $|V| = O(n)$ and $|E| = O(n + n^2 - nk)$, the result follows. □

5. Visibility calculations

The most difficult part of Algorithm 3.1 is to determine if S_j is visible from S_i, $j > i$. Let

$$|E| = e = \sum_{i=1}^{t} VL_i = O(n + n^2 - nk).$$

Our aim is to suggest an algorithm of $O(e)$ time for constructing the graph G of Algorithm 3.1. For this, we need a few basic concepts stated below.

An algorithm is given by Lee [5] for calculating visibility of a simple polygon P from an internal point u. It uses a stack and defines two kinds of blocking vertices. Suppose that we are looking at visibility from u. Look at three consecutive vertices v_{i-1}, v_i, v_{i+1} on bd(P) in counterclockwise order. Suppose $v_{i-1}v_iv_{i+1}$ takes a right turn and v_{i+1} is not visible from u (Fig. 6). Then, join uv_i and extend it to intersect bd(P) at some point $v_{i'}$. Then, obviously the region R bounded by the line $v_iv_{i'}$ and the portion of bd(P) from from v_i to $v_{i'}$ in counterclockwise direction direction (shaded in Fig. 6) is invisible from u. v_i is the lookahead vertex of u, denoted by LV(u), and the region R is the lookahead region of u, denoted by LR(u). Similarly, for left turns we define the backtracking vertex v_j of u, denoted by BV(u), and the backtracking region R' of u, denoted by BR(u) (Fig. 6).

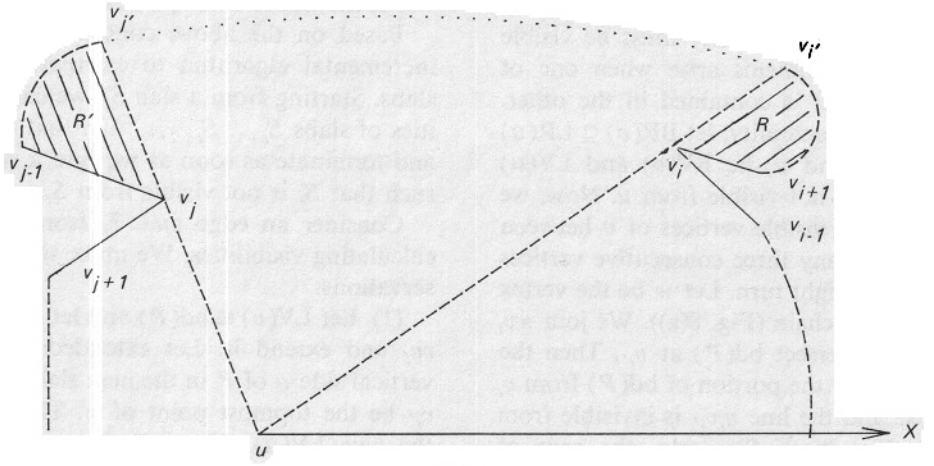

Fig. 6.

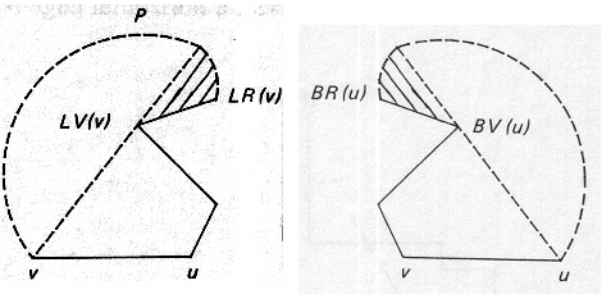

Fig. 7.

Now, consider visibility of a polygon P from an edge uv of P. Without any loss of generality, let uv be the bottommost edge and u be to the right of v. Then the following observations are true (Fig. 7):

(1) LR(v) and BR(u) are invisible from the edge vu;

(2) LR(u) and BR(v) may be partly or completely visible from the edge vu.

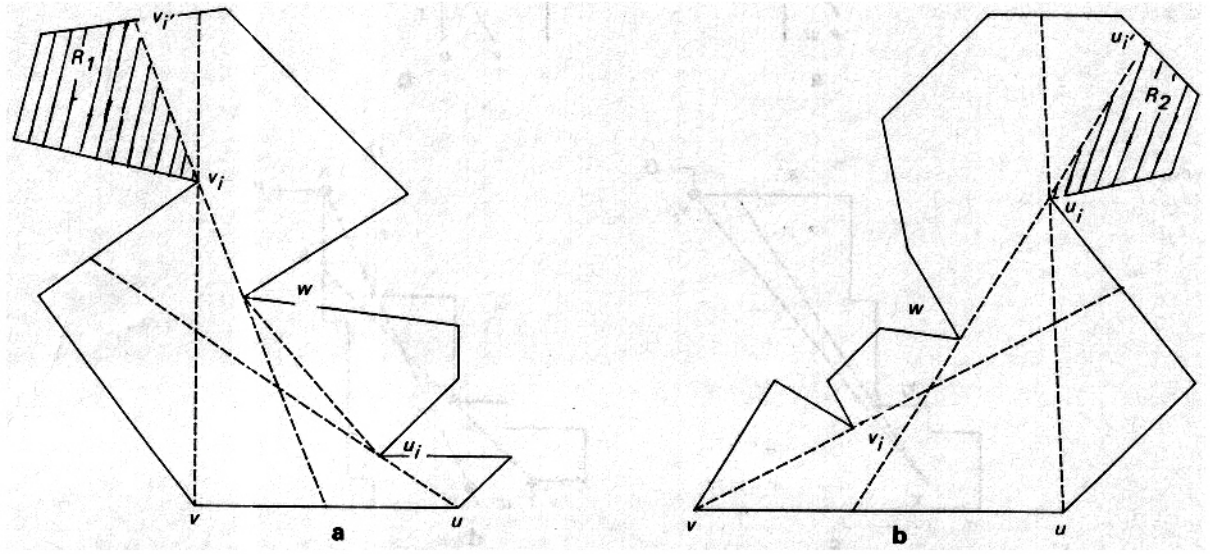

Fig. 8.

After removing LR(v) and BR(u), if LR(u) or BR(v) do not overlap, then they must be visible from the edge vu. Problems arise when one of them (or a part of it) is contained in the other. Without any loss of generality, let BR(v) $\subseteq$ LR(u) (Fig. 8(a)). Let v_i and u_i be BV(v) and LV(u) respectively. So v_i is not visible from u. Now, we obtain a chain of all visible vertices of v between u_i and v_i such that any three consecutive vertices in the chain form a right turn. Let w be the vertex preceeding v_i in the chain (Fig. 8(a)). We join wv_i and extend it to intersect bd(P) at $v_{i'}$. Then the region R_1 formed by the portion of bd(P) from v_i to $v_{i'}$ clockwise and the line $v_iv_{i'}$ is invisible from the edge vu (Fig. 8(a)). Similarly, the case of LR(u) $\subseteq$ BR(v) is illustrated in Fig. 8(b). In this case, the region R_2 is invisible from the edge vu.

All the above concepts are taken from [3].

Based on the above concepts, we propose an incremental algorithm to calculate visibilities of slabs. Starting from a slab S_i, we calculate visibilities of slabs S_{i+1}, S_{i+2}, ..., in that order from S_i and terminate as soon as we find a slab S_j, $j > i$, such that S_j is not visible from S_i.

Consider an edge $vu \subset T_i$ from which we are calculating visibilities. We make the following observations.

(1) Let LV(v) $\in$ bd(P) and let it be v_i. We join vv_i and extend it. Let extended vv_i intersect a vertical side a of P in the next slab (Fig. 9(a)). Let $v_{i'}$ be the topmost point of a. Then $v_{i'}$ becomes the new LV(u). Obviously, extended vv_i cannot intersect a horizontal edge of P before a. Otherwise, let extended vv_i intersect a horizontal edge b

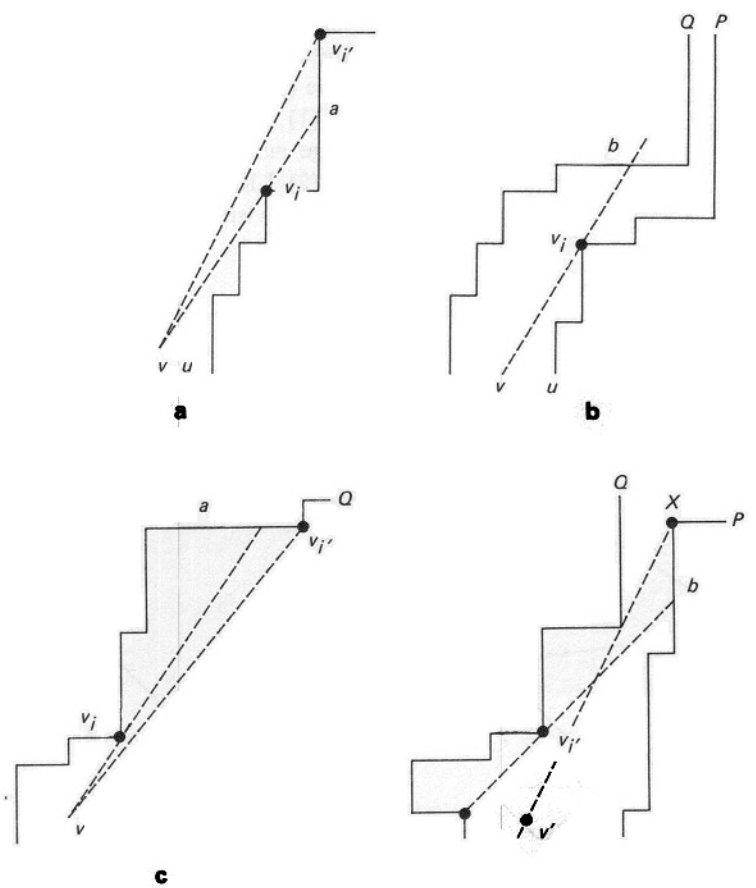

Fig. 9.

of Q (Fig. 9(b)), and let b belong to slab S_j. Then only slabs up to S_{j-1} are visible from edge vu. Obviously, extended vv_i cannot intersect a vertical edge of Q before b. When extended vv_i does not intersect an edge of P or Q in the next slab, v_i continues to be LV(v).

(2) Observations similar to (1) above are true for BR(u) also.

(3) Let BV(u) $\in$ bd(Q) and let it be v_i (Fig. 9(c)). Suppose extended vv_i intersects a horizontal edge a of Q and $v_{i'}$ is the rightmost endpoint of a (Fig. 9(c)). Then, $v_{i'}$ becomes new BV(v). Obviously, extended vv_i cannot intersect a vertical edge of Q before a. Otherwise, suppose extended vv_i intersects a vertical edge b of P (Fig. 9(d)). Let x be the topmost and rightmost corner point of the slab S_j to which the edge b belongs. Three cases are possible: the whole of LR(u) or a part of it is a subset of BR(v), or LR(u) and BR(v) are disjoint. Suppose the part of LR(u) which is a subset of BR(v) starts at slab S_k, $k \geq j$. Then we scan the edges of P and Q from S_k to S_j successively finding a chain of vertices visible with respect to bd(P) only such that any three consecutive vertices in the chain form a right turn. Note that whether vertices are visible from u with respect to bd(P) only can be calculated incrementally because since P and Q are both RRC polygons, we will never use a stack as in [5]. Let $v_{i'}$ be the vertex preceeding x in the chain. Then, join $v_{i'}x$ and extend it downwards to intersect vu at v' (Fig. 9(d)). Now we redefine v' as the new extreme point v and v'_i as the new BV(v). If LR(u) $\cap$ BR(v) $= \emptyset$, we take slab S_k as the slab containing v_i. However, for repeated applications of this step we will not start scanning from S_k every time as stated in observation (5) below, but only initially will we start from S_k.

(4) Observations similar to (3) above are true for LV(u) also.

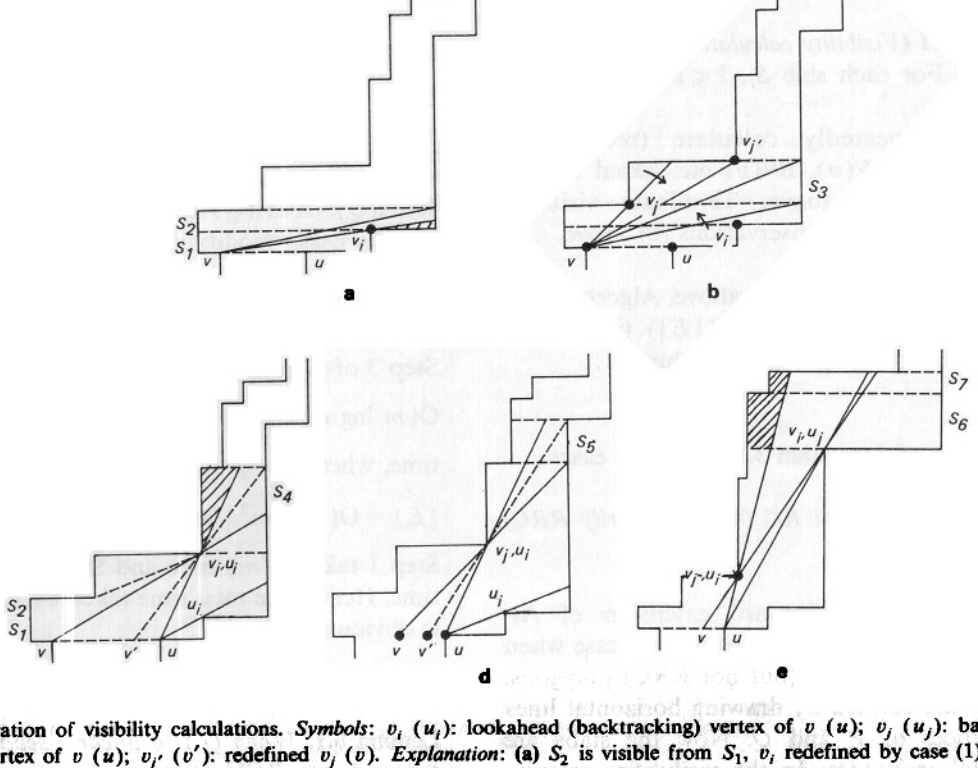

Fig. 10. Illustration of visibility calculations. *Symbols*: v_i (u_i): lookahead (backtracking) vertex of v (u); v_j (u_j): backtracking (lookahead) vertex of v (u); $v_{j'}$ (v'): redefined v_j (v). *Explanation*: **(a)** S_2 is visible from S_1, v_i redefined by case (1); **(b)** S_3 is visible from S_1, v_j redefined by case (3); **(c)** S_4 is visible from S_1, v redefined by case (3); **(d)** S_5 is visible from S_1, v redefined by case (3); **(e)** S_6 is visible from S_1, but S_7 is not visible from S_1 by case (1) and the algorithm stops.

(5) If observations (3) and (4) above are applied repeatedly, then the ith application of the particular observation scans the boundaries of P and Q, if necessary, starting from the edges of P and Q which were scanned last in the most recent previous application of the same observation. This is because since we are redefining u and v, the points on the boundaries of P and Q already scanned cannot become BV(v) or LV(u) afterwards.

(6) The repeated applications of the above observations must terminate either from observations (1) or (2) or from observations (3) or (4) when redefined u and v cross each other.

(7) The visibility calculation never examines more than O(VL_i) slabs from a slab S_i. Hence, in view of observation (5) above, the total time taken for visibility calculation of this slab is O(VL_i) only.

An example of repeated applications of the above observations for a slab is shown in Fig. 10.

We formally present the algorithm below.

Algorithm 5.1 (*Visibility calculations*)

Step 1: For each slab S_i, $1 \leqslant i \leqslant t$, a perform Step 2.

Step 2: Repeatedly calculate (recalculate) LV(u), LV(v), BV(u), BV(v) etc. based on observations (1)–(5) above. Terminate visibility calculations based on observations (6) above.

In view of observation (7) above, Algorithm 5.1 takes $O(\sum_{i=1}^{t} VL_i)$ time, i.e. O($|E|$) time, when $G = (V, E)$ is the graph of Algorithm 3.1.

6. Extension of Algorithm 3.1 to various cases

6.1. P and Q are both RC (not necessarily RRC) polygons

This is a straightforward extension of Algorithm 3.1. In Fig. 11 we consider the case when both P and Q are RC (but not RRC) polygons. The slabs are formed by drawing horizontal lines from vertices of P and Q. Now the slabs are trapezoidal in nature. In the visibilities calculations of Section 5, instead of vertical edges we may now have slanted edges, but the visibility calculation procedure remains similar if we treat the slanted edges in a way similar to vertical edges.

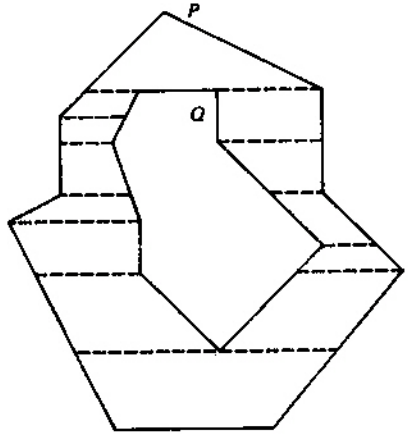

Fig. 11. Division of int(P) − int(Q) into slabs when P and Q are RC polygons.

6.2. P is RC and Q is an arbitrary simply polygon

By Proposition 4.2, for the minimal vertex nested polygon between P and Q is also the minimal vertex nested polygon between P and RCH(Q). RCH(Q) can be constructed in O(q) time by easily modifying the existing algorithm for computing the convex hull of a simple polygon as proposed by Lee [6] and also described in [8]. Let q' be the number of vertices in RCH(Q). So now Step 3 of Algorithm 3.1 takes

$$O(m \log m + |E|)$$

time, where $m = p + q'$ and

$$|E| = O(m + m^2 - mk).$$

Step 1 takes O(m) time and Step 2 takes O($|E|$) time. Hence, the total time taken by Algorithm 3.1 is obviously

$$O(n + m \log m + |E|).$$

Lemma 6.1. *When Q is a linearly separable monotone polygon*, RCH(Q) *has only a constant number of vertices on the average.*

Proof. Without any loss of generality, let the direction of monotonicity of Q be vertical, i.e. Q is a separable x-convex polygon. The boundary of Q can be decomposed into two chains, the right chain C_1 and the left chain C_2, such that C_1 and C_2 are monotone with respect to some vertical line and lie on the opposite sides of the line (Fig. 12). The vertices of Q having extreme y-coordinate values are assumed to belong to chain C_1 for concreteness. So, for an average analysis, one of the chains, say C_1, can be chosen arbitrarily over the whole real plane (as long as it is x-convex). Let l be the vertical line passing through the leftmost point in C_1 and l_1 and l_2 be horizontal lines passing through the vertices of Q with extreme y-coordinates (Fig. 12). Then, the chain C_2 may consist of points belonging to the region H of the real plane bounded by lines l_1, l_2 and l (shown shaded in Fig. 12). This selection of points on C_2 as above preserves the property of separability of the two chains.

For the average-case analysis, the following assumptions are made. The x-coordinates of points on the chains are chosen from the total real line (for chain C_1) or from a subset of the real line (for chain C_2) randomly and independently with uniform distribution. Furthermore, since the polygon Q is x-convex, chains C_1 or C_2 may consist of points $(x_1, y_1),\ldots,(x_r, y_r)$ in which the sequence $(y_1,\ldots, y_r)$, with each y_i is chosen from the total real line (for chain C_1) or from a subset of the real

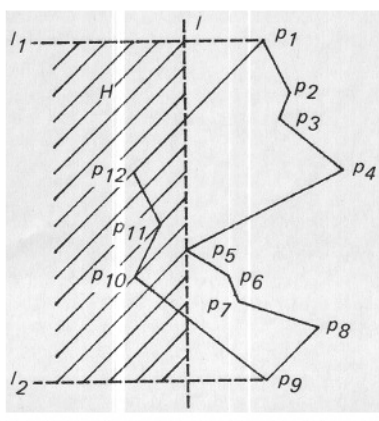

Fig. 12. Decomposition of a separable x-convex polygon into two chains C_1 and C_2. C_1 consists of points $(p_1,\ldots, p_9)$ and C_2 consists of points $(p_{10},\ldots, p_{14})$. p_5 is the leftmost point of C_1.

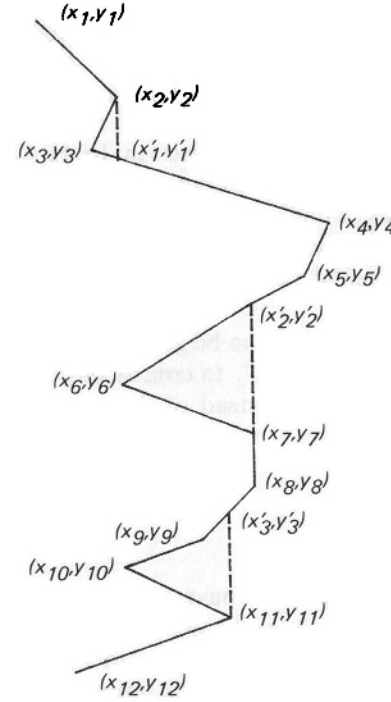

Fig. 13. RCH(C_1) has points (x_1, y_1), (x_2, y_2), (x_1', y_1'), (x_4, y_4), (x_5, y_5), (x_2', y_2'), (x_7, y_7), (x_8, y_8), (x_3', y_3'), (x_{11}, y_{11}), (x_{12}, y_{12}). x_1, x_2 and x_4 are the LR maxima, and x_{12}, x_{11}, x_8 and x_4 are the RL maxima of the sequence $(x_1,\ldots, x_{12})$. Three new points (x_1', y_1') (x_2', y_2') and (x_3', y_3') are introduced. The sequence $(y_1,\ldots, y_{12})$ is sorted for the x-convexity requirement.

line (for chain C_2), is necessarily sorted and a sequence of r unsorted y-coordinate values cannot form an x-convex chain. So we assume that each such sorted sequence of r y-coordinate values is equally likely to occur because we are interested in x-convex polygons only. We also assume that the x-coordinates of the points on the chain are chosen independently of each other.

First we prove that RCH(C_1) has only a constant number of points on the average. RCH(C_1) contains all s points (x_k, y_k) where x_k is an LR maximum or RL maximum of the sequence $(x_1,\ldots, x_r)$ [4] and possibly an additional set of at most s points $(x_1', y_1'),\ldots,(x_s', y_s')$ newly introduced (Fig. 13). Since $\frac{1}{2}$ is the probability that for any two randomly chosen numbers a and b, b is greater than a, the average number of LR maxima (or RL maxima) of the sequence

$(x_1, \ldots, x_r)$ is $1 + \frac{1}{2} + (\frac{1}{2})^2 + \cdots + (\frac{1}{2})^{r-1} = O(1)$. So, for a given set of r points $(x_1, y_1), \ldots, (x_r, y_r)$ in which the sequence $(y_1, \ldots, y_r)$ is fixed and sorted (due to the x-convexity requirement), the average number of vertices in $\mathrm{RCH}(C_1)$ is constant for every possible set of r x-coordinate value sequences. Finally, since all sorted y-coordinate sequences of length r are equally likely, the average number of vertices in $\mathrm{RCH}(C_1)$ remains constant.

A similar proof can be given for the other chain C_2 by considering C_2 to consist of points belonging to region H instead of the whole real plane. □

7. Conclusions

In this paper we have proposed approximate algorithms for special cases of the minimal vertex nested polygon problem. Further investigations may be carried out towards improving the nature of approximation of Algorithm 3.1 and to extend the algorithm for other types of polygons.

Acknowledgments

The authors wish to thank an anonymous referee for his helpful comments and suggestions during the initial review of this paper. The authors also wish to thank Mr. S.P. Pal and Mr. M.V. Ramana for their critical remarks during development of the algorithms.

References

[1] A. Aggarwal, H. Booth, J. O'Rourke, S. Suri and C.Y. Yap, Finding minimal convex nested polygons in: *Proc. 1st ACM Symp. on Computational Geometry* (1985) 296–304.

[2] M. Freedman and R. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, in: *Proc. 25th Ann. IEEE Symp. on Foundations of Computer Science* (1984) 338–346.

[3] S.K. Ghosh and R.K. Shyamsundar, A linear time algorithm for computing the polygon from an edge, in: *Proc. IEEE Conf. on Pattern Recognition and Image Processing* (1984).

[4] D.E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching* (Addison-Wesley, Reading, MA, 1973).

[5] D.T. Lee, Visibility of a simple polygon, *Comput. Vision, Graphics Image Process.* **22** (1983) 207–221.

[6] D.T. Lee, On finding the convex hull of a simple polygon, Tech. Report No. 80-03-FC-01, EE/CS Dept., Northwestern Univ.; also in: *Internat. J. Comput. Inform. Sci.* **12** (2) (1983) 87–98.

[7] S.R. Maddila and C.K. Yap, Moving a polygon around the corner in a corridor, in: *Proc. 2nd ACM Symp. on Computational Geometry* (1986) 187–192.

[8] F.P. Preparata and M.I. Shamos, *Computational Geometry — An Introduction* (Springer, Berlin, 1985).

[9] S.Suri and J. O'Rourke, Finding minimal nested polygons, Tech. Report, The Johns Hopkins Univ., Baltimore, MD, 1985.