Research Contribution

Monotone Polygon Containment Problems Under Translation

JUI-SHANG CHIU* and JIA-SHUNG WANG*

We investigate the problem of determining whether a polygon I can be translated to fit inside another polygon E without constructing the whole feasible region. For rectilinearly 2-concave polygons, an $O(m+n+k \log^2 mn)$ algorithm is presented in which m is the number of edges of I, n is the number of edges of E, and k is the number of sliding steps. In the worst case, k may be proportional to $O(mn)$. Since the feasible region may have $O(m^2n^2)$ edges, this algorithm runs more efficiently than one for finding the whole feasible region. An $O(m+n+k \log m+t)$ algorithm is also presented for monotone polygons. In the worst case, t may be proportional to $O(mn \alpha(mn) \log m)$, where $\alpha(\cdot)$ is the inverse of Ackermann's function.

1. Introduction

The polygon containment problem is to determine whether a polygon I can be translated to fit inside another polygon E. This problem has been studied by chazelle [5], Fortune [8], Baker *et al.* [4], Avnaim *et al.* [3], Martin *et al.* [12], and Ghosh [9]. For the case in which both polygons I and E are rectilinearly convex, Baker *et al.* [4] derived an algorithm that runs in $O(mn \log m)$ time where m is the number of edges of I and n is the number of edges of E. For rectilinear polygons, Avnaim *et al.* [3] proposed an $O(m^2n^2)$ algorithm that is worst-case optimal. Both algorithms find the whole feasible region. To the best of our knowledge, there is no easy way to modify these algorithms so that they stop whenever a feasible placement is found.

Designing an efficient algorithm to solve the polygon containment problem without constructing the whole boundary of the feasible region is an open problem in general. This paper presents a family of decision algorithms for problems in which (1) both polygons are rectilinearly convex, (2) both polygons are rectilinearly 2-concave, and (3) both polygons are monotone. Based on a plane-sweep-like approach, these algorithms terminate on the first feasible placement found, if one exists. Their complexity is dominated by the number of sliding steps, k, which depends not only on the number of edges of polygons but also on the size and the shape of polygons. Naturally, in most cases, this family of algorithms runs faster than those that find the whole feasible region.

*Institute of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30043, R.O.C.

2. Terminology

Given two polygons E and I in the plane, where E is fixed and I is mobile, we try to translate I to determine whether it can fit inside E. If it can, a feasible placement (or the whole feasible region) of I is reported. The placement of I is uniquely determined by the position of a specific point of I. This point is called the *reference point*. A *feasible placement* is a placement of the reference point such that I is contained in E. A *feasible region* is the set of all feasible placements of I. It can be the union of a finite number of polygons, line segments, and points.

A polygon P is *simple* if it has no holes and its edges are nonintersecting. P is *rectilinear* if it is simple and its bounding edges are either horizontal or vertical. P is *horizontally* (resp. *vertically*) *convex HC* (resp. *VC*) if it is rectilinear and its intersection with any horizontal (resp. vertical) line is either a single line segment or empty. P is *rectilinearly convex* if it is both HC and VC. Wood and Yap [16] give an alternative definition of convexity by quantifying the "degree of concavity." Let P be a rectilinear polygon represented by the vertices $(v_0, v_1, \ldots, v_{n-1}, v_n)$ on its contour, listed in clockwise order. In this list, $v_0=v_n$, we assign a *turning number* to each v_i such that

$$\tau_i = \begin{cases} 1 \text{ if } v_i \text{ is a left turn} \\ -1 \text{ if } v_i \text{ is a right turn.} \end{cases}$$

The *concavity* of P is defined to be

$$\max\left(\sum \tau_i, 0\right) \text{ for all sub-paths of } P.$$

A rectilinear polygon is *k-concave* if its concavity is k. According to this definition, 0-concave polygons are precisely rectangles, and 1-concave polygons are pre-

cisely rectilinearly convex polygons. Let HC_P (resp. VC_P) be the horizontally (resp. vertically) convex polygon containing P with the smallest area. It is observed that P is 2-concave if the only if the intersection of HC_P and VC_P is P.

A *chain* $C=(u_1, \ldots, u_n)$ is a planar straight-line graph with the vertex set $\{u_1, \ldots, u_n\}$ and the edge set $\{(u_i, u_{i+1}), i=1, \ldots, n-1\}$. C is said to be *monotone* with respect to a straight line l if any line orthogonal to l intersects C at no more than one point. A simple polygon is said to be monotone if its boundary can be decomposed into two chains that are monotone with respect to the same straight line l. A chain whose edges are either horizontal or vertical is said to be *X-monotone* (resp. *Y*-monotone) if its intersection with any line orthogonal to X-axis (resp. Y-axis) is either a single line segment, a single point, or empty. It is apparent that a rectilinearly convex polygon can be decomposed into two X-monotone (or Y-monotone) chains.

3. The Rectilinearly Convex Case

Assume two rectilinearly convex polygons I and E in the plane as shown in Fig. 1. I is contained in E if and only if all horizontal edges of I are inside E. The set of horizontal edges in the upper and lower X-monotone chains of I (resp. E) are denoted by $\mathcal{U}_I$ (resp. $\mathcal{U}_E$) and $\mathcal{L}_I$ (resp. $\mathcal{L}_E$), respectively. For a fixed placement of I, I is contained in E if and only if $\mathcal{U}_I$ lies below $\mathcal{U}_E$ and $\mathcal{L}_I$ lies beyond $\mathcal{L}_E$. Thus, we can separately compare $\mathcal{U}_I$ with $\mathcal{U}_E$ and $\mathcal{L}_I$ with $\mathcal{L}_E$.

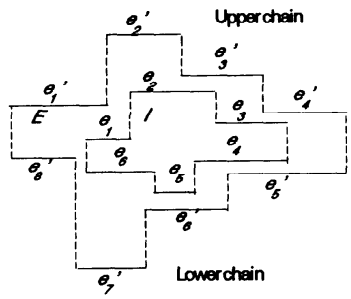

Fig. 1 An example of rectilinearly convex polygons.

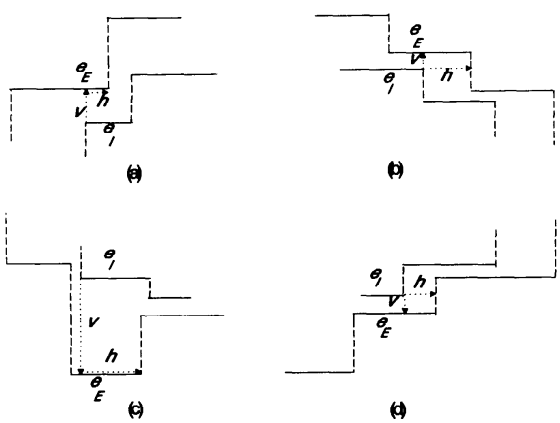

Fig. 2 The functions $v(e_I, e_E)$ and $h(e_I, e_E)$ for the hit pair (e_I, e_E).

First, we consider vertical translations of I(see Fig. 2). If I is slid upward (resp. downward), each edge in $\mathcal{U}_I$ (resp. $\mathcal{L}_I$) hits a set, called a *hit set*, of edges of $\mathcal{U}_E$ (resp. $\mathcal{L}_E$). We associate with each edge $e_I \in \mathcal{U}_I$ (resp. $\mathcal{L}_I$) a *hit pair* (e_I, e_E), where e_E is the edge with the lowest (resp. highest) height selected from the *hit set* of e_I. Note that only edges in the hit pairs are crucial for finding a vertical translation that will place I in the interior of E, and only one comparison is needed for each hit pair in order to determine whether such a vertical translation exists.

Next, we consider horizontal translations. As I is slid to the right for a sufficiently small distance, the hit pairs are unaffected. A placement of I at which the hit pairs are changed is called a *critical placement*. The horizontal step between two consecutive critical placement is called a *sliding step*. For each sliding step, exactly one hit pair will be changed. Note that a change of hit pair indicates the height that an edge of $\mathcal{U}_I$ (resp. $\mathcal{L}_I$) can move upward (resp. downward) before hitting an edge of E is changed. It is possible to translate I vertically to such a height that $\mathcal{L}_I$ lies beyond $\mathcal{L}_E$ and $\mathcal{U}_I$ remains below $\mathcal{U}_E$. Therefore, the decision of containment for this new placement should be reconsidered.

Let Q_I (resp. Q_E) be the smallest rectangle containing I (resp. E). We may slide I from where the lower left corners of Q_I and Q_E are overlapped to where the lower right corners of Q_I and Q_E are overlapped. Since each edge of I may hit a total of $O(n)$ edges of E, there are $O(mn)$ critical placements in the worst case. A straightforward approach to determining the containment is to test all the critical placements, but this takes $O(m^2n)$ time. However, by associating the functions $v(e_I, e_E)$ and $h(e_I, e_E)$ for each hit pair (e_I, e_E) as follows, we show how to adapt the *plane-sweep* strategy to run in $O(mn \log m)$ time.

$$v(e_I, e_E)=\begin{cases} e_E.y-e_I.y & \text{if } e_I \in \mathcal{U}_I, \\ e_I.y-e_E.y & \text{if } e_I \in \mathcal{L}_I; \end{cases}$$

$$h(e_I, e_E)=\begin{cases} e_E.\text{right.x}-e_I.\text{right.x} \\ \quad \text{if } e_E \in \text{the right part of chain}, \\ e_E.\text{right.x}-e_I.\text{left.x} \\ \quad \text{otherwise}; \end{cases}$$

where $e_E.y$ ($e_I.y$) is the y-coordinate of e_E (e_I), e_E.right.x (e_I.right.x) is the x-coordinate of the right endpoint of e_E (e_I), and e_I.left.x is the x-coordinate of the left endpoint of e_I. The value $v(e_I, e_E)$ represents the maximum distance that e_I can move upward (or downward) without hitting its corresponding edge e_E, and the value

$h(e_I, e_E)$ gives the minimum distance that e_I must move to the right to change its own hit pair.

We also define the following three guiding measures:

$$v_{\mathcal{U}} = \min \{v(e_I, e_E) \mid e_I \in \mathcal{U}_I\},$$

$$v_{\mathcal{L}} = \min \{v(e_I, e_E) \mid e_I \in \mathcal{L}_I\},$$

$$h = \min \{h(e_I, e_E) \mid e_I \in \mathcal{U}_I \text{ or } e_I \in \mathcal{L}_I\}.$$

The value $v_{\mathcal{U}}$ represents the maximum height that I can move upward such that $\mathcal{U}_I$ remains below $\mathcal{U}_E$. The value $-v_{\mathcal{L}}$ represents the minimum height that I must move upward such that $\mathcal{L}_I$ lies beyond $\mathcal{L}_E$. Finally, the value h represents the distance that I can move to the right until a hit pair is changed. It is not hard to verify that a feasible placement exists if and only if $v_{\mathcal{U}} \geq -v_{\mathcal{L}}$. Hence, by computing the value of $v_{\mathcal{U}}$ and $v_{\mathcal{L}}$ we can determine whether I can be translated vertically to fit into E. Similarly, we can determine the distance for the next sliding step by computing the value of h.

By organizing the information on these hit pairs into priority queues [1], we can adapt the plane-sweep strategy efficiently. While I slides from the left to the right step by step, we can determine the feasible placement by computing the guiding measures. The process is terminated as soon as the first feasible placement is found and reported.

The algorithm is as follows:

Step 1. Decompose both polygons I and E into two X-monotone chains respectively;

Step 2. Let I slide step by step from the left to the right. In each sliding step, we compute the values $v_{\mathcal{U}}$ (or $v_{\mathcal{L}}$) and h. Whenever $v_{\mathcal{U}} \geq -v_{\mathcal{L}}$, a feasible placement is reported and the process is terminated successfully.

Here, we analyze the performance of the above algorithm. In each sliding step, we need to update the distance h and the height $v_{\mathcal{U}}$ or $v_{\mathcal{L}}$. We keep track of these values by using three priority queues. $O(\log m)$ time is needed both for extracting and for updating the minimum values of $v_{\mathcal{U}}$ (or $v_{\mathcal{L}}$) and h. Thus, the construction and maintenance of the data structures in Step 2 takes $O(k \log m)$ time, where k is the number of sliding steps before a feasible placement is found. As for Step 1, the time needed to decompose I and E into two chains each is $O(m+n)$, assuming that all the edges in I and E are given in clockwise order. Therefore, the complexity of the above algorithm is $O(m+n+k \log m)$.

We have shown that there are $O(mn)$ sliding steps in the worst case. Hence, the worst-case time complexity is $O(mn \log m)$. This is the same that of as the algorithm proposed by Baker *et al.* [4]. However, ours is capable of finding the first feasible placement and then stopping early. In contrast, theirs cannot report the answer before the whole feasible region has been found. Since the feasible region may have $O(mn)$ edges, our algorithm runs faster in most cases.

To find the whole feasible region, it suffices to report the feasible region within each sliding step and then to slide until the lower right corner of Q_I and Q_E are overlapped. The total time needed is still $O(mn \log m)$.

Our algorithm is not applicable when both I and e are not rectilinearly convex, because the edges of the concave part become obstracles, so the polygons I and E cannot be decomposed into two X-monotone chains each. However, if both polygons E and I are VC or HC, our algorithm still works after a slight modification of the distance function $h(\cdot, \cdot)$.

4. The Rectilinearly 2-Concave Case

Turning to the case in which both polygons I and E are rectilinearly 2-concave, we present an efficient algorithm to determine whether E can contain I. This algorithm runs in $O(m+n+k \log^2 mn)$ time, where k is the number of sliding steps before a feasible placement is found. In the worst case, k may be proportional to $O(mn)$. We also present a similar algorithm for finding the whole feasible region. The time complexity becomes $O(mn \log mn + t)$, where t is the number of edges of the whole feasible region. In the worst case, t may be proportional to $O(m^2n^2)$.

A 2-concave polygon P can be regarded as the result of intersection by one horizontally convex polygon HC_P and one vertically convex polygon VC_P (see Fig. 3). Suppose that we can transform I into HC_I and VC_I, and E into HC_E and VC_E. Note that I can fit inside HC_E (resp. VC_E) if and only if HC_I (resp. VC_I) can be contained in HC_E (resp. VC_E). Thus, the feasible region of I inside E

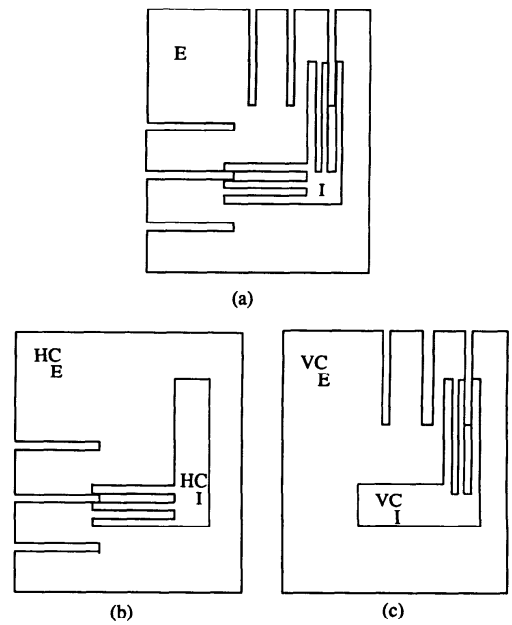

Fig. 3 (a) The rectilinearly 2-concave case, (b) The horizontally convex case, and (c) The vertically convex case.

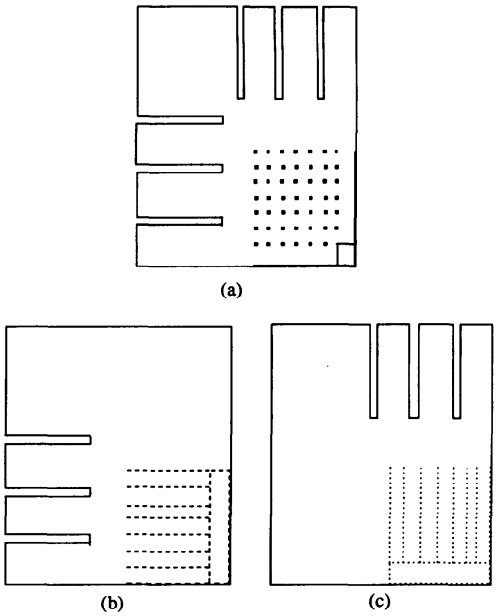

Fig. 4 The feasible regions.

is exactly the intersection of the feasible region HC_I inside HC_E and the feasible region VC_I inside VC_E. Fig. 4 shows a typical example.

To solve the containment problem of 2-concave polygons, we first attempt to solve the problem in both the horizontally convex and vertically convex cases. We have proposed an algorithm for solving these two cases in Section 3. The remaining problems are: (1) how to transform a 2-concave polygon into one HC and one VC, (2) how to find the whole feasible region, and (3) how to determine the containment as soon as possible.

Nicholl *et al.* [14] proposed a linear time algorithm to obtain a minimum-area rectilinearly convex polygon that contains the given rectilinear polygon. This algorithm can also be used to transform a 2-concave polygon P into one HC_P and one VC_P in linear time.

As polygons I and E are transformed, the feasible regions for the HC case and the VC case can be found independently. Note that the feasible region in each case is regarded as a union of feasible rectangles. There are at most $O(mn)$ feasible rectangles in both cases. Thus, finding the feasible region becomes a problem of computing the intersection of two sets of feasible rectangles. This can be done in $O(mn \log mn + t)$ time [15], where t is the number of edges of the feasible region. Therefore, the total time needed for computing the whole feasible region is $O(mn \log mn + t)$. Note that t is negligible if we simply want to determine whether E can contain I. This can also be determined by applying the priority search tree [13].

However, to determine the containment as soon as possible, neither the HC case nor the VC case is necessary for finding the whole feasible region. We may slide HC_I and VC_I to search for a feasible rectangle step by step alternately. Whenever a feasible rectangle for the HC (resp. VC) case is found, we begin to examine whether it intersects with any currently found feasible rectangles for VC_I (resp. HC_I). The process is terminated as soon as a nonempty intersection is found.

To take advantage of answering rectangle queries quickly, we apply the dynamic data structures proposed by Edelsbrunner [6, 7]. Thus, the rectangle insertion and the query of rectangle intersection both need $O(\log^2 mn)$ time. Therefore, the total time needed for finding a feasible placement is $O(m+n+k \log^2 mn)$, where k is the number of sliding steps. This algorithm runs faster in most cases, though it is inferior in its worst-case time complexity.

The algorithm is as follows:

Step 1. Transform polygons I and E into HC_I, VC_I, HC_E, and VC_E respectively.

Step 2. Slide VC_I horizontally until a new feasible rectangle is found. If one is not found, report that no feasible placement is possible and terminate the process.

Step 3. Insert the new rectangle into the set of found feasible rectangles $\mathscr{R}_{VC}$, and then check whether it intersects with the set of found feasible rectangles $\mathscr{R}_{HC}$. If so, report it and terminate the process.

Step 4. Slide HC_I vertically until a new feasible rectangle is found. If one is not found, report that no feasible placement is possible and terminate the process.

Step 5. Insert the new rectangle into the set of found feasible rectangles $\mathscr{R}_{HC}$, and then check whether it intersects with the set of found feasible rectangles $\mathscr{R}_{VC}$. If so, report this solution and terminate the process.

Step 6. Repeat steps 2 to 5.

In the 2-concave case, we have shown that there indeed exists a more efficient algorithm that can determine the containment without constructing the whole feasible region.

5. The Montone Case

A monotone polygon can be decomposed into two monotone chains. Consider the case shown in Fig. 5. To determine the containment, as in the rectilinearly convex case, we may associate each edge of I with a hit pair and then determine the containment by using these pairs. However, unlike in the rectilinearly convex case, the height for a hit pair is not a constant again, since it changes continuously as I slides to the right.

Consider Fig. 6. For a given hit pair (e_I, e_E), when edge e_I of $\mathscr{U}_I$ (resp. $\mathscr{L}_I$) moves upward (resp. downward) to a height of $v(e_I, e_E)$, it will touch a point on edge e_E of $\mathscr{U}_E$ (resp. $\mathscr{L}_E$). We denote this as the *contact point*. Contact points can be classified into two different categories: (1) (*edge contact*) one of the endpoints of e_I lies on e_E (2) (*vertex contact*) one of the endpoints of e_E lies on e_I.

Consider three different cases of edge contact as

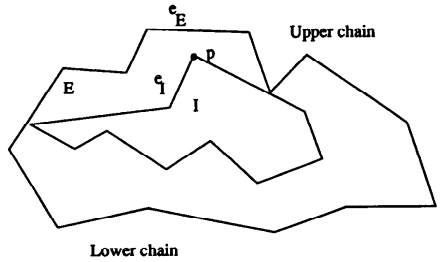

Fig. 5 An example of monotone polygons.

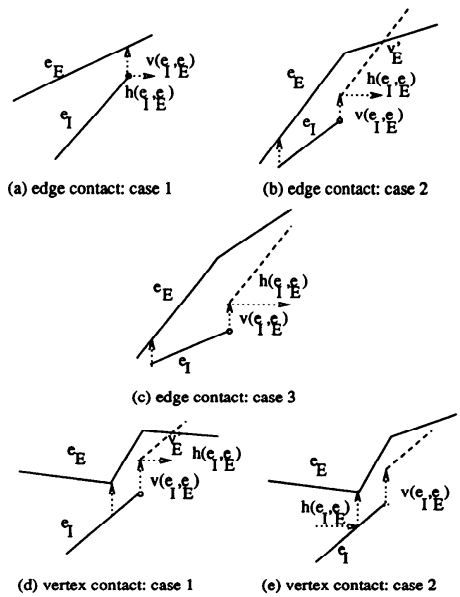

Fig. 6 Illustration of five situations for the edge contact and vertex contact.

shown in Fig. 6(a)-(c). Imagine that e_I, starting from a contact, slides rightward along e_E; an endpoint v_I of e_I will keep contact with e_E until e_I hits another edge e'_E. Of course, the hit pair (e_I, e_E) is changed to (e_I, e'_E) as we have already discussed for the rectilinear convex case. Here, for the monotone case, we define $v(e_I, e_E)$ as the vertical distance between endpoint v_I and edge e_E, $h(e_I, e_E)$ as the horizontal distance that e_I can slide along e_E to the right until it hits e'_E, and $s(e_I, e_E)$ as the slope at which e_I must slide to maintain contact with e_E. Note that we do not really slide I up and down, but instead, slide I to the right step by step.

Next, consider two cases of vertex contact, as shown in Fig. 6(d) and 6(e). Imagine that e_I, starting from a contact, slides rightward passing through v_E; e_I will keep contact with v_E until it hits e'_E.

Let a straight line containing edge e be formulated by an equation $y=a(e)\cdot x+b(e)$. The values of $s(e_I, e_E)$, $v(e_I, e_E)$, and $h(e_I, e_E)$ for the hit pair (e_I, e_E) can be computed as follows:

(1) edge contact (v_I, e_E):

$$s(e_I, e_E)=a(e_E);$$

$$v(e_I, e_E)=\begin{cases} a(e_E)\cdot e_I.\text{right.x}+b(e_E)-e_I.\text{right.y} \\ \quad \text{if } a(e_I)\geq a(e_E), \\ a(e_E)\cdot e_I.\text{left.x}+b(e_E)-e_I.\text{left.y} \\ \quad \text{if } a(e_I)<(e_E); \end{cases}$$

$$h(e_I, e_E)=\begin{cases} e_E.\text{right.x}-e_I.\text{right.x} \\ \quad \text{if } a(e_I)\geq a(e_E), \\ v'_E.\text{x}-e_I.\text{right.x} \\ \quad \text{if there exists an edge } e'_E \text{ on the right of } e_E \\ \quad \text{intersecting } e_I \text{ while we slide } e_I \text{ along } e_E, \\ e_E.\text{right.x}-e_I.\text{left.x} \\ \quad \text{otherwise.} \end{cases}$$

(2) vertex contact (e_I, e_E):

$$s(e_I, e_E)=a(e_I);$$

$$v(e_I, e_E)=v_E.y-(a(e_I)\cdot v_E.\text{x}+b(e_I));$$

$$h(e_I, e_E)=\begin{cases} v'_E.\text{x}-e_I.\text{right.x} \\ \quad \text{if there exists an edge } e'_E \text{ on the right of } e_E \\ \quad \text{intersecting } e_I \text{ at a point } v'_E \text{ on it while we} \\ \quad \text{slide } e_I, \text{ keeping contact with } v_E, \\ v_E.\text{x}-e_I.\text{left.x} \\ \quad \text{otherwise.} \end{cases}$$

In the following paragraphs, we employ the concept of contact chain and envelope to describe the necessary conditions for containment. Consider the edge e_I in Fig. 7. While I is moving to the right in such a way that some edges of E are always in contact with e_I, the locus of the reference point of I forms a chain, which we call the *contact chain* of e_I. Note that this contact chain preserves the monotone property, and may consist of at most n line segments. In general, there may be m contact chains for all of the edges of I. For convenience' sake, we denote the set of upper contact chains by $\mathscr{C}_\mathscr{U}$ and the set of lower contact chains by $\mathscr{C}_\mathscr{L}$.

From the above discussion, we know that the upper (or lower) contact chain of e_I contributes one necessary

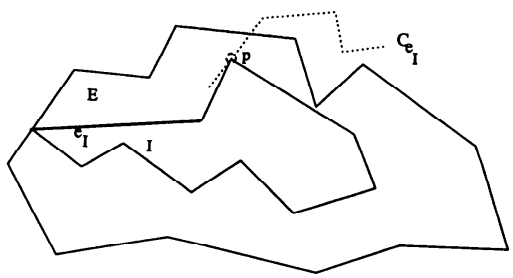

Fig. 7 The contact chain.

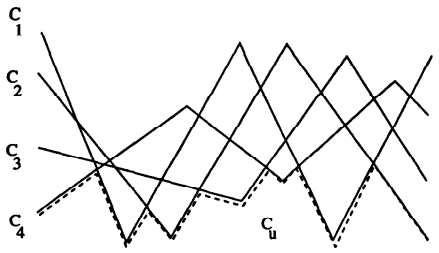

Fig. 8 The lower envelope $C_{\#}$ of a set of upper contact chains.

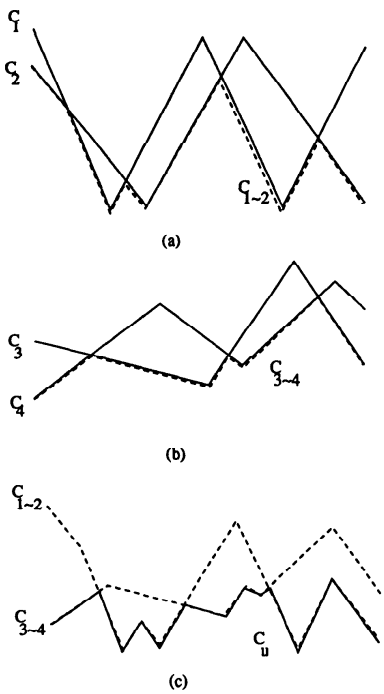

Fig. 9 A divide-and-conquer approach to computer $C_{\#}$.

condition for the feasible placement of I. That is, the reference point p_I of I should be located below (resp. above) the upper (resp. lower) contact chain of e_I. Thus, for a feasible placement of I, the reference point p_I must be located below all of the upper contact chains and above all of the lower contact chains. These will be all the necessary conditions for solving this containment problem. We denote the intersection of all the lower (resp. upper) regions bounded by $\mathscr{C}_{\#}$ (resp. $\mathscr{C}_{\mathscr{S}}$) as the *lower* (resp. *upper*) *envelope* $C_{\#}$ (resp. $C_{\mathscr{S}}$). See Fig. 8.

We also adapt the plane-sweep approach in order to solve the polygon containment problem of the monotone case. In each sliding step, the distance $h(\cdot,\cdot)$, height $v(\cdot,\cdot)$ and slope $s(\cdot,\cdot)$ for an appropriate hit pair are computed and then a new edge of an associated contact chain is generated. After all the contact chains have

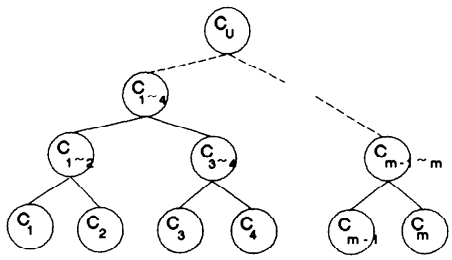

Fig. 10 A binary merge tree for the envelope of contact chains.

been generated, the lower envelope $C_{\#}$ and the upper envelope $C_{\mathscr{S}}$ can be obtained. Finally, the entire feasible region is found by simply computing the intersection of $C_{\#}$ and $C_{\mathscr{S}}$.

The remaining problem is to find the envelope of m contact chains efficiently. Several authors have observed that a straightforward divide-and-conquer algorithm can solve a similar problem of finding the upper envelope of n line segements in $O(n\,\alpha(n)\log n)$ time [2, 10], where $\alpha(n)$ is the inverse of Ackermann's function. Note that the upper envelope of n line segments may consist of $O(n\,\alpha(n))$ edges [10]. Since our problem has m contact chains, each of them with at most n line segments, the envelope of these contact chains can be computed in $O(mn\,\alpha(mn)\log m)$ time, using the same approach (see Fig. 9). Recently, an optimal algorithm was proposed by Hershberger [11]. Hence, $C_{\#}$ and $C_{\mathscr{S}}$ can be obtained in $O(mn\log mn)$ time by Hershberger's method.

In addition, the intersection of $C_{\#}$ and $C_{\mathscr{S}}$ can be computed in $O(mn\,\alpha(mn))$ time. Also, as in the rectilinearly convex case, there are $O(mn)$ sliding steps, each which takes $O(\log m)$ time. Therefore, the worst-case complexity of our algorithm is $O(mn\log mn)$.

However, the above algorithm is not our final goal, since we are interested in solving the decision problem, namely, to find a feasible placement as soon as possible. Under the circumstances, some modifications are necessary for computing $C_{\#}$ and $C_{\mathscr{S}}$ step by step. To combine the divide-and-conquer approach with the plane-sweep approach, we employ a data structure called a *binary merge tree* to help the process. The tree is shown in Fig. 10. There are m contact chains stored in m leaves, respectively. Initially, each chain may contain only one edge. As I is sliding to the right, a new edge of some contact chain, say C_i, is generated in one sliding step. Then, we start the merging process leading from the leaf containing C_i to the root. Two contact chains are merged into a new chain, which is the envelope of these two chains. For instance, $C_{1\sim2}$ is the envelope of C_1 and C_2. Thus, the current $C_{\#}$ (resp. $C_{\mathscr{S}}$) is found in the root. Note that the merging process is performed in such a way that only those edges that belong to the envelope of two chains are produced. We shall discuss this in the next paragraph. The above steps are repeated until a nonempty intersection region between $C_{\#}$ and

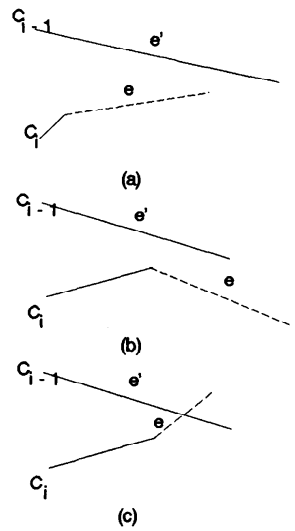

Fig. 11 Three cases in the merging process of two chains.

$C_{\mathscr{S}}$ is found.

Consider the case shown in Fig. 11(a). The currently generated edge e of the contact chain C_i lies below the edge e' of the contact chain C_{i-1}, and the right endpoint of e located to the left of the right endpoint of e'. At this moment, it is not certain whether the whole edge of e' will become part of the upper envelope of C_{i-1} and C_i or not. Thus, the merging process should be delayed until either of the two cases shown in Fig. 11(b) and 11(c) appears. In the former case, the whole edge of e' is part of the upper envelope. In the latter case, only some parts of e and e' are in the upper envelope. In accordance with the above discussion, to ensure that the merging process always produces edges that belong to the envelope of two chains, the merging process is advanced to a higher level of the tree only if the current envelope needs to be updated.

Finally, we analyze the performance of the above algorithm. In each sliding step, a new edge of an associated contact chain is generated and the envelopes along a path of the tree from this leaf to the root should be updated to generated some edges of $C_{\mathscr{U}}$ (or $C_{\mathscr{V}}$). Since both trees are balanced, the tree height is $O(\log m)$, the updates take at most $O(\log m+t)$ time, where t is the number of edges updated along the path. Therefore, the complexity of the above algorithm is $O(m+n+k \log m+t)$, where k is the number of sliding steps. In the worst case, the total number of edges updated before finding a feasible placement is

$$\begin{cases} t(m, n)=2t(m/2)+O(mn\ \alpha(mn)), & \text{if } m \geq 2, \\ t(1, n)=O(n) & \text{Otherwise,} \end{cases}$$

which is $O(mn\ \alpha(mn) \log m)$.

Here, we should point out that the subproblem of finding the envelope of a set of chains has several special features, that is, (1) it forms m monotone chains each of which has $O(n)$ line segements, (2), there are $O(m+n)$ slopes of the segments, and (3) these chains are similar to each other. It is not known whether the complexity of the envelope can be reduced from $O(mn\ \alpha(mn))$ to $O(mn\ \alpha(m))$ or even $O(mn)$, but the probability is high.

6. Discussion

If a rectilinear polygon I can fit inside a rectilinearly convex polygon E, the rectilinearly convex hull I' of I can also be contained in E. Since I' can be computed in linear time [14], we can use our method to solve this polygon containment problem. For a similar reason, if I is rectilinear and E is rectilinearly 2-concave, I can be considered as a rectilinearly 2-concave polygon. Furthermore, if I is simple and E is monotone, I can be considered as monotone. Moreover, if some edges of I (or E) are circular arcs and preserve the monotone property, the algorithm for the monotone case may still work. However, some modifications are needed, for instance, to compute the distances and to compute the intersections between arcs and line segments.

In this paper, we have presented a family of algorithms for the restricted cases in which two polygons are rectilinearly convex, 2-concave, and monotone. The complexity of these algorithms is dominated by the number of sliding steps, which depends not only on the number of edges of the polygons but also on the size and the shape of the polygons. In these restricted cases, we have shown that it is possible to develop a more efficient algorithm that finds a feasible placement without constructing the whole feasible region. However, in more general cases in which both polygons are k-concave, where $k>2$, the problem remains open.

References

1. AHO, A. V., HOPCROFT, J. E. and ULLMAN, J. D. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1975.
2. ATALLAH, M. J. Dynamic Computational Geometry, *IEEE 24th Annual Symposium on Foundations of Computer Science* (1983), 92-99.
3. AVNAIM, F. and BOISSONNAT, J. D. Simultaneous Containment of Several Polygons, *3rd ACM Symposium on Computational Geometry* (1987), 242-250.
4. BAKER, B. S., FORTUNE, S. J. and MAHANEY, S. R. Polygon Containment under Translation, *Journal of Algorithms*, 7 (1986), 532-548.
5. CHAZELLE, B. The Polygon Containment Problem, *Advances in Computing Research*, 1 (1983), 1-33, JAI Press.
6. EDELSBRUNNER, H. *Dynamic Data Structures for Orthogonal Intersection Queries*, Rep. F59. Tech. Univ., Graz, Institute für Informations-verarbeitung, 1980.
7. EDELSBRUNNER, H. and MAURER, H. A. On the Intersection of Orthogonal Objects, *Inf. Process. Lett.*, 13 (1981), 177-181.
8. FORTUNE, S. J. A Fast Algorithm for Polygon Containment by Translation, *Automata, Language, and Programming*, 12th Colloquium, in Lecture Notes in Computer Science 194, Springer-Verlag, New York (1985), 180-198.

9. GHOSH, P. K. A Solution of Polygon Containment, Spatial Planning, and Other Related Problems Using Minkowski Operations, *Computer Vision, Graphics, and image Processing*, **49** (1990), 1-35.
10. HART, S. and SHARIR, M. Nonlinearity of Davenport-Schinzel Sequence and of Generalized Path Compression Schemes, *Combinatorica*, **6** (1986), 151-177.
11. HERSHBERGER, J. Finding the Upper Envelope of n Line Segments in $O(n \log n)$ Time, *Inf. Process. Lett.*, **33** (1989), 169-174.
12. MARTIN, R. R. and STEPHENSON, P. C. Putting Objects into Boxes, *Computer Aided Design*, **20** (1988), 506-514.
13. MCCREIGHT, E. M. Priority Search Trees, *SIAM J. Comput.*, **14** (1985), 257-276.
14. NICHOLL, T. M., LEE, D. T., LIAO, Y. Z. and WONG, C. K. Constructing the X-Y Convex Hull of a Set of X-Y Polygons, *BIT*, **23** (1983), 456-471.
15. WIDMAYER, P. and WOOD, D. A Time- and Space-Optimal Algorithm for Boolean Mask Operations for Orthogonal Polygons, *Computer Vision, Graphics and Image Processing*, **41** (1988), 14-27.
16. WOOD, D. and YAP, C. K. The Orthogonal Convex Skull Problem, *Discrete Computational Geometry*, **3** (1988), 349-365.

(Received February 7, 1990; revised June 11, 1990)