アルゴリズム 12－18
(1989. 11. 21)

Monotone Polygon Containment Problems Under Translation

Jui-Shang Chiu and Jia-Shung Wang

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

Abstract

We investigate the problem of determining whether a polygon I can be translated to fit inside another polygon E without constructing the whole boundary of the feasible region. In the case of rectilinearly 2-concave polygons, an $O(mn \log^2 mn)$ algorithm is presented where m is the number of edges of I and n is the number of edges of E. Since the feasible region may have $O(m^2n^2)$ edges, this algorithm is more efficiently to solve the problem. Also, an $O(mn \log m)$ algorithm is presented to solve the case of monotone polygons.

1. Introduction

The polygon containment problem is the problem of determining whether a polygon I can be translated to fit inside another polygon E. This problem has been studied by Chazelle [4], Fortune [6], Baker *et al* [3], Avnaim *et al* [2], and Martin *et al* [7]. In the case that both polygons I and E are rectilinearly convex, Baker *et al* [3] derived an algorithm that runs in time $O(mn \log m)$ where m is the number of edges of I and n is the number of edges of E. And, in the case of rectilinear polygons, Avnaim *et al* [2] proposed an $O(m^2n^2)$ algorithm which is worst-case optimal. Both algorithms find the whole feasible region.

Unfortunately, it remains open whether we can design a more efficient algorithm in general that determines the relationship of containment without constructing the whole boundary of the feasible region. This paper presents a family of decision algorithms for the problems that (1) both polygons are rectilinearly convex, (2) both polygons are rectilinearly 2-concave, and (3) both polygon are monotone. These algorithms will terminate on the first-found feasible placement if there exists one. Naturally, in most of cases, the family of algorithms will run faster than the one that finds the whole feasible region.

2. Terminology

Given two polygons E and I in the plane, E is fixed and I is mobile. We try to translate I to determine whether I can fit inside E. If yes, report a feasible placement (or the whole feasible region) of I. A placement of I is uniquely determined by a position of a specific point of I. This point is called the *reference point*. A *feasible placement* is a placement of I 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 hole and its edges are nonintersecting. P is *rectilinear* if it is simple and its bounding edges are either horizontal or vertical. P is *horizontally (vertically) convex HC (VC)* if it is rectilinear and its intersection with any horizontal (vertical) line is either a single line segment or empty. P is *rectilinearly convex* (or *1-concave* [10]) if it is both HC and VC. P is *2-concave* if there exist one HC and one VC such that the intersecion of these two polygons is P.

A *chain* $C = (u_1,...,u_n)$ is a planar straight-line graph with vertex set $\{u_1,...,u_n\}$ and edge set $\{(u_i,u_{i+1}), i=1,...,n-1\}$. C is said to be *monotone* with respect to a straight line l if any line orthogonal to l intersects C in at most one point. A simple polygon is said to be monotone if its boundary can be decomposed into two chains monotone with respect to the same straight line l. A chain whose edges are either horizontal or vertical is said to be *X-monotone* (*Y-monotone*) if its intersection with any line orthogonal

to X–axis (Y–axis) is either a single line segment or empty. It is appearance that a rectilinearly convex polygon can be decomposed into two X–monotone (or Y–monotone) chains.

3. The Rectilinearly Convex Case

Given two rectilinearly convex polygons I and E in the plane as shown in Fig. 1. Clearly, I is contained in E if all horizontal edges of I are inside of E. The set of horizontal edges in the upper and lower X–monotone chains of I (E) are denoted by $\mathcal{U}_I$ ($\mathcal{U}_E$) and $\mathcal{L}_I$ ($\mathcal{L}_E$) respectively. To determine the relationship of containment for a placement of I, the comparisons can be done between $\mathcal{U}_I$ and $\mathcal{U}_E$ and between $\mathcal{L}_I$ and $\mathcal{L}_E$ independently. That is, I is contained in E if $\mathcal{U}_I$ lies *below* $\mathcal{U}_E$ and $\mathcal{L}_I$ lies *beyond* $\mathcal{L}_E$. However, each edge in $\mathcal{U}_I$ ($\mathcal{L}_I$) will hit a set of corresponding edges in $\mathcal{U}_E$ ($\mathcal{L}_E$) if I is sliding up (or down). We associate each edge $e_I \in \mathcal{U}_I$ ($\mathcal{L}_I$) a *hit pair* (e_I, e_E), where e_E is an edge with lowest (highest) height selected from the *hit set* of e_I. That is, e_E will be the first edge in the set hit by e_I. Note that only those edges in the hit pairs will affect the relationship of containment. Thus, only one comparison is needed for each hit pair to determine that relationship.

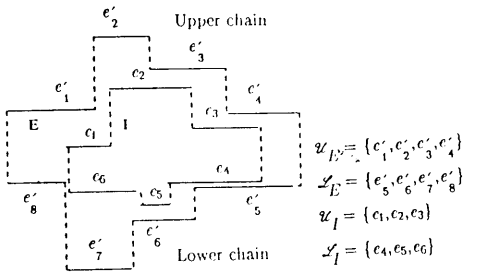

Fig. 1. An example of rectilinearly convex polygons.

As I slides to right for a small distance, the hit pairs are unaffected. However, as I slides to right for a certain distance, one edge of I will change its corresponding edge in the hit pair. For such a discrete distance we called a *sliding step*. For each sliding step, exactly one hit pair will be changed. Note that a change of hit pair represents that the height an edge can move upward (or downward) before hitting its corresponding edge is also changed. It is possible to translate I vertically for such a height that $\mathcal{L}_I$ can lie beyond $\mathcal{L}_E$ and $\mathcal{U}_I$ can lie below $\mathcal{U}_E$. Therefore, the relationship of containment for the new placement should be reconsidered.

Since each edge of I may hit $O(n)$ edges of E totally in the trip of I, there are at most $O(mn)$ sliding steps concerning with the hit pair changes. Hence, there are $O(mn)$ critical placements in the worst case. A straightforward approach to determine the relationship of containment is to test all the critical placements. This will cost $O(m^2n)$ time. However, it is possible to associate two functions for a hit pair (e_I, e_E), one is about what *height* edge e_I can move upward (or downward) before touching edge e_E, and the other is about what *distance* edge e_I can slide to the right before changing its hit pair. Note that for a feasible placement of I, e_I should not be higher than e_E if $e_I \in \mathcal{U}_I$, and e_I should not be lower than e_E if $e_I \in \mathcal{L}_I$. Hence, these two functions can be used as guidances for the movements of I in a given placement. The height function can be used to determine a feasible placement and the distance function can be used to determine the distance for one sliding step.

The functions $v(e_I, e_E)$ and $h(e_I, e_E)$ for a hit pair (e_I, e_E) are defined as follows:

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

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

where $e_E.y$ ($e_I.y$) is the y–coordinate of e_E (e_I), $e_E.\text{right.x}$ ($e_I.\text{right.x}$) is the x–coordinate of the right endpoint of e_E (e_I), and $e_I.\text{left.x}$ is the x–coordinate of the left endpoint of e_I.

The value $v(e_I, e_E)$ represents the maximum distance such 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 to the right such that e_I will change its own hit pair. See Fig. 2.

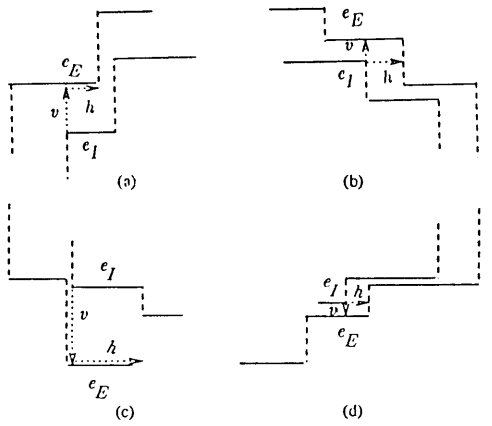

Fig. 2. The functions $v(e_I, e_E)$ and $h(e_I, e_E)$ for the hit pair (e_I, e_E).

We also define three guiding measures as below:

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

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

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

The value $v_{\mathcal{U}}$ represents the maximum height I can move upward such that $\mathcal{U}_I$ can lie below $\mathcal{U}_E$. The value $-v_{\mathcal{L}}$ represents the minimum height I must move upward such that $\mathcal{L}_I$ can lie beyond $\mathcal{L}_E$. And, the value h represents the distance I can move to right such that exactly one hit pair will be changed. It is not hard to verify that there exists a feasible placement if and only if $v_{\mathcal{U}} \geq -v_{\mathcal{L}}$. Hence, by $v_{\mathcal{U}}$ and $v_{\mathcal{L}}$ one determines whether I can be translated vertically to fit into E, and by h one determines the distance for such a sliding step.

By organizing these information of 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 will terminate as soon as the first feasible placement is found and reported.

We now describe the algorithm 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 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, where Q_I (Q_E) is the smallest rectangle containing I (E). And 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 worst–case performance of the above algorithm. We have showed that there are $O(mn)$ sliding steps in the worst case. 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. So, to extract the minimum values of $v_{\mathcal{U}}$ (or $v_{\mathcal{L}}$) and h, and to update their values need $O(\log m)$. Thus, the time needs to construct and maintain the data structures in Step 2 takes $O(mn \log m)$ time in the worst case. As for Step 1, the time needed to decompose I and E each into two chains respectively can be done in $O(m+n)$ steps assuming all of the edges in I and E are given in the clockwise order. Therefore, the worst–case complexity of the above algorithm is $O(mn \log m)$.

Though, in the worst case, the time complexity of the above algorithm is the same as Baker *et al*'s [3]. However, the former is capable of finding the first feasible placement and then stop early. In contrast to this, the latter can not report the answer before the whole feasible region has been found. Since the feasible region may have $O(mn)$ edges, our algorithm runs faster for the most of cases.

To find the whole feasible region, the additional works are to slide every step and to report the feasible region within each sliding step. The total time is still $O(mn \log m)$.

If I and E are not rectilinearly convex, then the algorithm may fail. Because the edges of concave part become obstacles so that the polygons I and E can not be decomposed into two X–monotone chains respectively. However, if both polygons E and I are VC or HC, our algorithm still succeed after a slight modifi-

cation on the distance function $h(\cdot,\cdot)$.

4. The Rectilinearly 2–Concave Case

We now consider the case that both polygons I and E are rectilinearly 2–concave. An efficient algorithm to determine whether E contains I is presented. This algorithm runs in time $O(mn \log^2 mn)$. Also, a similar algorithm is presented to find the whole feasible region. The time complexity becomes $O(mn \log mn + k)$, where k is the number of edges of the whole feasible region. In the worst case, k may proportion to $O(m^2n^2)$.

It is observed that 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 we can transform I into HC_I and VC_I, and E into HC_E and VC_E. It's not hard to see that the feasible region of I inside E is exactly the intersection of feasible region HC_I inside HC_E and VC_I inside VC_E. Fig. 4 depicts a typical example.

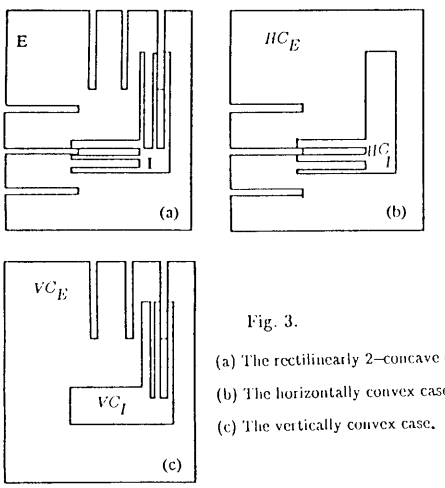

Fig. 3.
(a) The rectilinearly 2–concave case.
(b) The horizontally convex case.
(c) The vertically convex case.

To solve the containment problem of 2–concave polygons, we first turn to solve the problem in both the horizontally convex and vertically convex cases. We have proposed an algorithm to solve 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 relationship of containment as soon as possible.

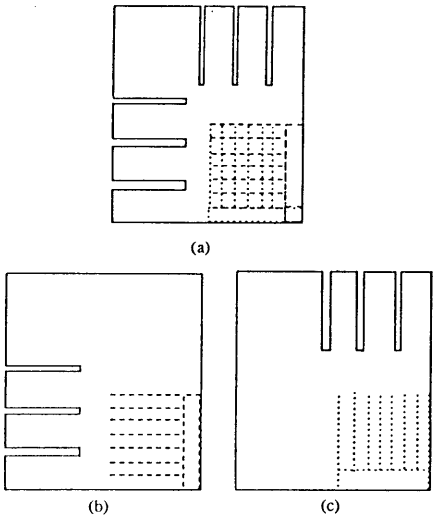

Fig. 4. The feasible regions.

Nicholl, *et al.* [8] proposed a linear time algorithm to obtain a minimum area rectilinearly convex polygon which contains the given rectilinear polygon. This algorithm can be also 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 placements for the HC case and the VC case can be found independently. Note that the feasible region in each case can be regarded as a union of feasible rectangles. And there exists at most $O(mn)$ feasible rectangles in both cases. Because we have known that the whole feasible region of I inside E can be considered as the intersection of two feasible regions obtain in the HC and VC cases respectively. Thus, to find the feasible region becomes to compute the intersection of two sets of feasible rectangles. This can be done in time $O(mn \log mn + k)$ [9], where k is the number of edges of the feasible region. Therefore, the total time for computing the whole feasible region takes $O(mn \log mn + k)$.

However, if we simply want to determine whether E contains I, naturally it is not necessary to find the whole feasible region for both the VC and HC cases. We may slide HC_I and VC_I to search for a feasible

rectangle alternatively. And as a feasible rectangle for HC (VC) case is found, we begin to examine whether it intersects with any found feasible rectangles in the VC (HC) case or not. The process is terminated as soon as there exists an nonempty intersection being found.

To take the advantage of answering rectangle queries, we apply the dynamic data structures proposed by Edelsbrunner [5]. Thus, the rectangle insertion needs $O(\log^2 mn)$, and the query of rectangle intersection takes $O(\log^2 mn + e)$ time where e is the number of edges being intersected. Therefore, the total time for finding a feasible placement needs $O(mn \log^2 mn)$ in the worst case.

The algorithm is described 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 there is a new feasible rectangle found. Otherwise, report that no feasible placement is possible and terminate the process.

Step 3. Insert the new one into the set of found feasible rectangles $\mathcal{R}_{VC}$, and then examine whether it intersects with the set of found feasible rectangles $\mathcal{R}_{HC}$. If the answer is yes, report it and terminate the process.

Step 4. Slide HC_I vertically until there is a new feasible rectangle found. Otherwise, report that no feasible placement is possible and terminate the process.

Step 5. Insert the new one into the set of found feasible rectangles $\mathcal{R}_{HC}$, and then examine whether it intersects with the set of found feasible rectangles $\mathcal{R}_{VC}$. If the answer is true, report this solution and terminate the process.

Step 6. Repeat step 2 to step 5 again.

In the 2–concave case, we showed that there indeed exists a more efficient algorithm that determines the relationship of containment without constructing the whole feasible region.

5. The Monotone Case

A monotone polygon can be decomposed into two monotone chains. Consider the case as shown in Fig. 5. To determine the relationship of containment, as the rectilinearly convex case, we may associate each edge of I a hit pair and then compare the edges within the hit pairs. However, one comparison of height between a hit pair is insufficient, since the height changes continuously as I slides to the right.

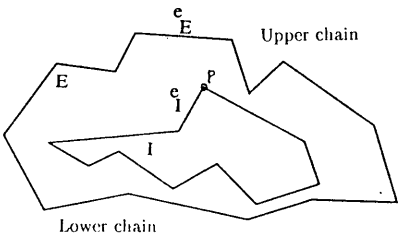

Fig. 5. An example of monotone polygons.

For a given hit pair (e_I, e_E), when edge e_I of $\mathcal{U}_I$ ($\mathcal{L}_I$) moves upward (downward) for height $v(e_I, e_E)$, it will hit edge e_E of $\mathcal{U}_E$ ($\mathcal{L}_E$) and generate a contact point. We say that these two edges e_I and e_E generate a contact point if one of the following two situations occurs: (1) either one endpoint of e_I lies on e_E (*edge contact*), or (2) one endpoint of e_E lies on e_I (*vertex contact*). See Fig. 6.

Consider an edge contact induced by an endpoint v_I of e_I and the edge e_E as if e_I has been translated vertically. Imagine that e_I, starting from this contact, slides rightward along e_E, v_I will keep contact with e_E until e_I hits another edge e'_E. And then hit pair (e_I, e_E) is changed to (e_I, e'_E). We define $v(e_I, e_E)$ to be the vertical distance between endpoint v_I and edge e_E, $h(e_I, e_E)$ to be the horizontal distance such that edge e_I can slide along e_E to the right until another contact point is generated, and $s(e_I, e_E)$ to be the slope of such a sliding of e_I that keeps contact with e_E. Note that we do not really slide I up and down, but instead I is slid to the right step by step.

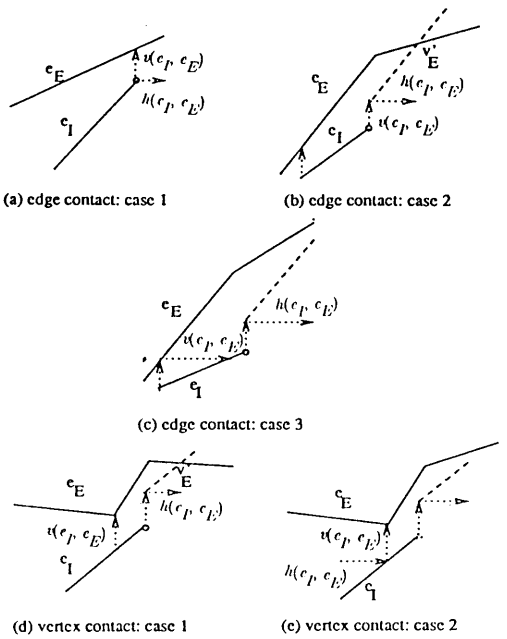

Fig. 6. Illustration of five situations for the edge contact and vertex contact.

Now, consider a vertex contact induced by the edge e_I and an endpoint v_E of e_E as if e_I has been moved upward for the height $v(e_I, e_E)$. Imagine that e_I starting from this contact, slides rightward passing thru v_E. e_I will keep contact with v_E until e_I hits another edge e'_E at the distance $h(e_I, e_E)$ on the right of 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):

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

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

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

(2) vertex contact (e_I, v_E):

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

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

$$h(e_I, e_E) = \begin{cases} v'_E.\text{x} - e_I.\text{right.x} \\ \quad \text{if there exists an edge } e'_E \text{ on the} \\ \quad \text{right of } e_E \text{ intersects } e_I \text{ on } v'_E \\ \quad \text{while } e_I \text{ is sliding thru } v_E, \\ v_E.\text{x} - e_I.\text{left.x} \\ \quad \text{otherwise.} \end{cases}$$

If I is moving to the right in such a way that there always exist some edges in E keeping contact with edge e_I, the locus of the reference point of I will form a *contact chain*. Note that this contact chain preserves the monotone property. It may consist of at most n line segments. Each edge of I is associated with a contact chain. We denote the set of upper contact chains by $\mathscr{C}_{\mathscr{U}}$ and the set of lower contact chains by $\mathscr{C}_{\mathscr{L}}$. See Fig. 7.

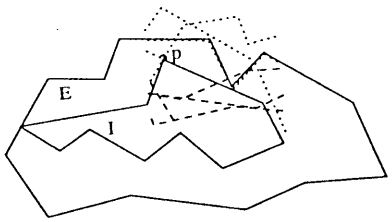

Fig. 7. The upper contact chains (dotted lines) and the lower contact chains (broken lines).

The upper (lower) contact chain of e_I gives some information about the feasible placement of I. That is, the reference point p_I of I should be located below (beyond) the upper (lower) contact chain of e_I. In

general, for a feasible placement of I, the reference point p_I must lie below all of the upper contact chains and beyond all of the lower contact chains. Let $C_{\mathcal{U}}$ be the intersection of all the lower regions bounded by $\mathcal{C}_{\mathcal{U}}$ and $C_{\mathcal{L}}$ be the intersection of all the upper regions bounded by $\mathcal{C}_{\mathcal{L}}$. Clearly, there exists a feasible placement if and only if the intersection of $C_{\mathcal{U}}$ and $C_{\mathcal{L}}$ is not empty.

Note that the number of intersections between the upper (lower) contact chains may proportion to $O(m^2n)$ in the worst case. However, the boundaries of $C_{\mathcal{U}}$ and $C_{\mathcal{L}}$ may consist of at most $O(mn)$ edges. In order to compute $C_{\mathcal{U}}$ and $C_{\mathcal{L}}$ efficiently, the intersections of chains between $\mathcal{C}_{\mathcal{U}}$ ($\mathcal{C}_{\mathcal{L}}$) that do not contribute to $C_{\mathcal{U}}$ ($C_{\mathcal{L}}$) should be discarded.

Fortunately, this subproblem can be solved efficiently using the *divide–and–conquer* strategy. Consider Fig. 8. There are four chains, C_1, C_2, C_3 and C_4 in $\mathcal{C}_{\mathcal{U}}$. To compute $C_{\mathcal{U}}$, we first divide $\mathcal{C}_{\mathcal{U}}$ into two sets $\{C_1, C_2\}$ and $\{C_3, C_4\}$, then compute the intersection $C_{1\text{-}2}$ of C_1 and C_2, and the intersection $C_{3\text{-}4}$ of C_3 and C_4 individually. The boundary for the intersection region still form a monotone chain with possible doubled number of edges. Finally, we compute the intersection region of $C_{1\text{-}2}$ and $C_{3\text{-}4}$, that is $C_{\mathcal{U}}$. See Fig. 9. It is not hard to see that, in general, the number of edges manipulated by this divide–and–conquer process is $O(mn \log m)$.

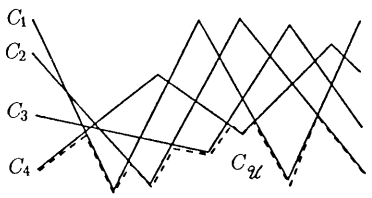

Fig. 8. The number of intersections between $\mathcal{C}_{\mathcal{U}}$ is an magnitude of order over the number of edges of $C_{\mathcal{U}}$

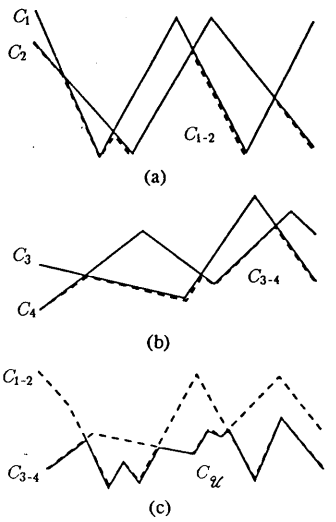

Fig. 9. A divide–and–conquer approach to compute $C_{\mathcal{U}}$

According to the above discussions, we adopt a data structure, *binary merge tree*, to help the process. In the leaves of this tree, we store all the upper (lower) contact chains. Two chains are then merged into a new chain as their parent node. The merging process is performed from bottom to top. Thus $C_{\mathcal{U}}$ ($C_{\mathcal{L}}$) can be obtained in the root of tree. See Fig. 10.

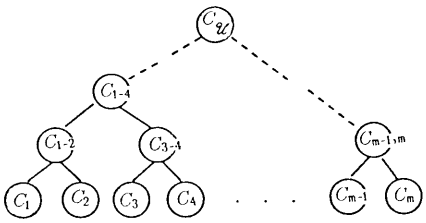

Fig. 10. A binary merge tree for the upper contact chains.

To solve the monotone polygon containment problem efficiently, we also use the planesweep approach. The edges of upper (lower) contact chains are generated serially. Initially each chain may contain at most one edge. As I is sliding to the right, the next edge of the contact chains is generated step by step. Since only one new edge in a upper (or lower) contact chain is generated in a sliding step, only a path in the

binary merge tree should be updated. This path is specified by the new updating chain to the root. After a new edge of $C_{\mathcal{U}}$ ($C_{\mathcal{L}}$) is generated, the intersection between $C_{\mathcal{U}}$ and $C_{\mathcal{L}}$ within the sliding step can be computed. The process is terminated as soon as there exists an nonempty intersection being found.

Here, we analyze the worst–case performance of the above algorithm. There are $O(mn)$ sliding steps. In each sliding step, we need to compute the distance $h(\cdot,\cdot)$, height $v(\cdot,\cdot)$ and slope $s(\cdot,\cdot)$ for some hit pair and generate a new edge of some contact chain. Then a path of the tree from this chain to the root should be updated to generate an edge of $C_{\mathcal{U}}$ or $C_{\mathcal{L}}$. Since there are totally m leaves in two trees and both are balanced, the updates along the path cost at most $O(\log m)$ time. Therefore, the worst–case complexity is $O(mn \log m)$.

Similarly, to find the whole feasible region, the additional work is to report the intersection region found between $C_{\mathcal{U}}$ and $C_{\mathcal{L}}$ in every step. The total time is still $O(mn \log m)$.

6. Discussions

If a rectilinear polygon I can fit inside a rectilinearly convex polygon E, there must exists a rectilinearly convex polygon I' which contains I and is contained in E. Since I' can be computed in linear time [8], so we can deal exclusively with rectilinearly convex polygons. 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 (E) are circular arcs and preserve the monotone property, the algorithm for the monotone case may still work. However, some modifications are needed such as to compute the distance and to compute the intersection between arcs and line segments.

We have presented a family of algorithms for the restricted cases in which both polygons are rectilinearly convex, 2–concave and monotone. The complexity of these algorithms depend not only on the number of edges of polygons but also on the size and shape. And in these restricted cases we showed that it is possible to develop a more efficient algorithm that determines the relationship of containment without constructing the whole feasible region. However, in the cases that both polygons are k–concave where $k > 2$, the problem remains open.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, Mass., 1975.

[2] F. Avnaim and J. D. Boissonnat, "Simultaneous Containment of Several Polygons," *3rd ACM Symposium on Computational Geometry*, pp. 242–250, 1987.

[3] B. S. Baker, and S. J. Fortune, and S. R. Mahaney, "Polygon Containment under Translation", *Journal of Algorithms*, Vol. 7, pp. 532–548, 1986.

[4] B. Chazelle, "The Polygon Containment Problem," *Advances in Computing Research*, Vol. 1, pp. 1–33, JAI Press, 1983.

[5] H. Edelsbrunner, *Dynamic Data Structures for Orthogonal Intersection Queries*, Rep. F59. Tech. Univ. Graz, Institute für Informationsverarbeitung, 1980.

[6] S. J. Fortune, "A Fast Algorithm for Polygon Containment by Translation," *Autometa, Language, and Programming*, 12th Colliquium, in Lecture Notes in Computer Science 194, Springer–Verlag, New York, pp. 180–198, 1985.

[7] R. R. Martin and P. C. Stephenson, "Putting Objects into Boxes", *Computer Aided Design*, Vol. 20, pp. 506–514, 1988.

[8] T. M. Nicholl, D. T. Lee, Y. Z. Liao and C. K. Wong, "Constructing the X–Y Convex Hull of a Set of X–Y Polygons,", *BIT*, Vol. 23, pp. 456–471, 1983.

[9] P. Widmayer and D. Wood, "A Time– and Space–Optimal Algorithm for Boolean Mask Operations for Orthogonal Polygons," *Computer Vision, Graphics and Image Processing*, Vol. 41, pp. 14–27, 1988.

[10] D. Wood and C. K. Yap, "The Orthogonal Convex Skull Problem," *Discrete Computational Geometry*, Vol. 3, pp. 349–365, 1988.