Finding Largest Rectangle Inside a Digital Object

Apurba Sarkar[1](✉), Arindam Biswas[2], Mousumi Dutt[3], and Arnab Bhattacharya[4]

[1] Department of Computer Science and Technology, Indian Institute of Engineering Science and Technology, Howrah, India
as.besu@gmail.com

[2] Department of Information Technology, Indian Institute of Engineering Science and Technology, Howrah, India
barindam@gmail.com

[3] Department of Computer Science and Engineering, International Institute of Information Technology, Naya Raipur, India
duttmousumi@gmail.com

[4] Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India
arnabb@iitk.ac.in

Abstract. We present a combinatorial algorithm which runs in $O(n \log n)$ time to find largest rectangle (LR) inside a given digital object without holes, n being the number of pixels on the contour of digital object. The object is imposed on background isothetic grid and inner isothetic cover is obtained for a particular grid size, g, which tightly inscribes the digital object. Certain combinatorial rules are applied on the isothetic cover to obtain the largest rectangle. The largest rectangle is useful for shape analysis of digital objects by varying grid size, by rotating the object, etc. Experimental results on different digital objects are also presented.

Keywords: Digital object · Isothetic grid · Rectangle · Inner isothetic cover · Shape analysis

1 Introduction

The problem of finding the Largest area axis-parallel Rectangle (LR) inside a general polygon of n vertices is a geometric optimization problem in the class of polygon inclusion problem [4]. There are many solutions for this problem in various scenarios (e.g. in convex polygon, in orthogonal polygon, etc.) because of the practical importance of the problem. Chazelle et al. [5,6] proposed an algorithm to find largest area rectangle with sides parallel to the given rectangle containing n points and reported that their algorithm runs in $O(n \log^3 n)$ time and $O(n \log n)$ space. Aggarwal et al. [1] simplifies that algorithm by Chazelle et al. [5,6] and proposed an algorithm that takes same $O(n \log^3 n)$ time but

© Springer International Publishing Switzerland 2016
A. Bac and J.-L. Mari (Eds.): CTIC 2016, LNCS 9667, pp. 157–169, 2016.
DOI: 10.1007/978-3-319-39441-1_15

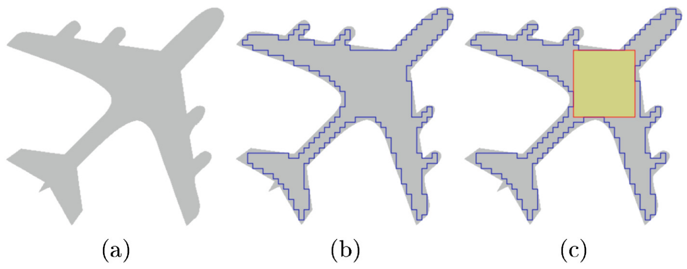


Fig. 1. (a) The digital object, A, (b) Inner isothetic cover ($g = 8$), (c) Largest Rectangle.

$O(n)$ space. They proposed another algorithm that runs in $O(n \log^2 n)$ time and $O(n)$ space. Daniels et al. [7] considered a geometric optimization problem of finding maximum area axis parallel rectangle from a n-vertex general polygon. They characterized the largest area rectangle problem by considering different cases based on the types of contacts between the rectangle and the polygon. They also proposed a framework that can transform an algorithm for orthogonal polygons into an algorithm for non-orthogonal polygons and showed that the running time of their algorithm for general polygons to be $O(n \log^2 n)$. They have established lower bound for finding the largest empty rectangles in both self-intersecting polygons and general polygons with holes to be $O(n \log n)$. McKenna et al. [9] use a divide-and-conquer approach to find the LR in an orthogonal polygon in $O(n \log^5 n)$ time. For the merge step at the first level of divide-and-conquer, they obtain an orthogonal, vertically separated, horizontally convex polygon. At the second level, their merge step produces an orthogonally convex polygon, for which they solve the LR problem in $O(n \log^3 n)$ time. They also establish a lower bound of time in $\Omega(n \log n)$ for finding the LR in orthogonal polygons with degenerate holes, which implies the same lower bound for general polygons with degenerate holes.

LR problem has many applications in electronic design automation, design and verification of physical layout of integrated circuits [10,11]. Largest area rectangle problem has many interesting industrial applications also, e.g., consider a sheet of fabric or a rectangular piece metal with certain number of flaws in it. This problem can be salvaged to find a maximum area rectangular sheet that does not contain any flaws.

In this paper we present another flavor of the same problem - finding largest rectangle in a digital object which is useful for shape analysis of the object. It is to be noted here that the resulting largest rectangle may not be unique. The digital object (Fig. 1(a)) is imposed on a background grid (grid size may vary depending on the shape and size of the object). Inner isothetic cover which tightly inscribes the digital object is shown in Fig. 1(b). The corresponding largest rectangle is shown in Fig. 1(c) for grid size $g = 8$. In Sect. 2 required definitions and procedure to obtain inner isothetic cover are explained in brief. While traversing along the inner isothetic cover combinatorial rules are applied to obtain the largest rectangle. The algorithm presented in this paper runs in $O(n \log n)$ time, where n is the number of pixels on the contour of the digital object. In Sect. 3, the procedure to obtain largest rectangle is stated in details including rules, algorithm, time complexity, and demonstration. Experimental results are given in Sect. 4 to verify the algorithm. Section 5 contains concluding remarks.

2 Definitions and Preliminaries

A digital object A is a 8-connected component [8]. The *background grid* is given by $\mathbb{G} = (\mathbb{H}, \mathbb{V})$, where $\mathbb{H}$ and $\mathbb{V}$ represent the respective sets of (equi-spaced) horizontal grid lines and vertical grid lines. The *grid size* g is defined as the distance between two consecutive horizontal/vertical grid lines. A *grid point* is the point of intersection of a horizontal and a vertical grid line. A *unit grid block* (UGB) is the smallest square having its four vertices as four grid points and edges as grid edges. An *isothetic polygon* P is a simple polygon (i.e., with non-intersecting sides) of finite size in $\mathbb{Z}^2$ whose alternate sides are subsets of the members of $\mathbb{H}$ and $\mathbb{V}$. The polygon P, hence given by a finite set of UGBs, is represented by the (ordered) sequence of its vertices, which are grid points. The border B_P of P is the set of points belonging to its sides. The interior of P is the set of points in the union of its constituting UGBs excluding the border of P. An isothetic cover has two type of vertices 90^0 (type **1**) and 270^0 (type **3**). The *inner (isothetic) cover* (IIC), denoted by $P(A)$, is a set of inner polygons and (inner) hole polygons, such that the region, given by the union of the inner polygons minus the union of the interiors of the hole polygons, contains a UGB if and only if it is a subset of A. An edge of P defined by two consecutive vertices of type **1** is termed as a *convex edge*, as it gives rise to a *convexity*. Similarly, an edge defined by two consecutive type **3** vertices gives rise to a *concavity*, and hence termed as a *concave edge*.

Using the algorithm in [2,3], we obtain (the ordered set of vertices of) P for A, which is, therefore, the maximum-area isothetic polygon inscribing A. During the construction of P, the vertices and grid points lying on the edges of P are dynamically inserted in a circular doubly-linked list, L, and simultaneously in two lexicographically sorted (in increasing order) lists, L_x and L_y, with respective primary and secondary keys as x- and y-coordinates for L_x, and opposite for L_y. Each node of the list L has two level pointers, the lower level pointers are used to link both edge and corner points of IIC and the top level pointers are used to link only the corner points of IIC.

3 Procedure to Determine Largest Rectangle

To find a largest rectangle, the inner isothetic cover P (constructed using algorithm in [2,3]) is traversed in anti-clockwise direction from its top left corner. During this traversal, whenever a convex edge is encountered, corresponding histogram polygon (i.e., a portion of the main polygon) is constructed, as explained in Sect. 3.1. Largest rectangle inscribed in the histogram polygon

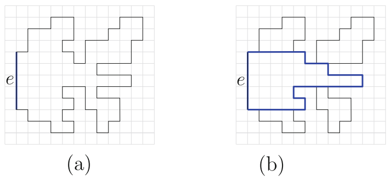


Fig. 2. Histogram polygon w.r.t. a convex edge.

is determined (Sect. 3.2). The convexity encountered in P is reduced using the appropriate reduction rule explained in Sect. 3.3. After reduction, it may give rise to a convex edge, corresponding to which the histogram polygon, thereof the largest rectangle inscribed in it, is determined. Thus, the traversal continues till it reaches the start point of P, the largest of all such largest rectangles, inscribed in the histogram polygons of convex edges, is the resulting largest rectangle of P.

3.1 Finding Histogram Polygon

During traversal, whenever a convex edge (say e) is detected, histogram polygon is considered, whose base is e. The base may lie horizontally (at top or bottom) or vertically (at left or right). The base is extended to extract the histogram polygon which does not contain any concavity horizontally (vertically) if the base is vertical (horizontal). Figure 2(a) shows a vertical convex edge e at left side, which is extended upto the right side of P in such a way that there is no horizontal concavities. Corresponding histogram polygon is shown in Fig. 2(b) in blue outline.

3.2 Finding Largest Rectangle in a Histogram Polygon

To find the largest rectangle in histogram polygon, the opposite side of the base, e, is traversed. Whenever a convex edge is encountered, the area of the corresponding rectangle is determined, which is compared with the stored largest rectangle (or global largest rectangle). Larger rectangle is stored as global largest rectangle. After considering the rectangle in histogram polygon, P is reduced based on the reduction rules stated in Sect. 3.3. This process continues until it traverses all the convex edges opposite to e. For example, in Fig. 3(a), the area corresponding to the first convexity, shown as the gray rectangle, is computed as $l_1 \times l_2$ and the convexity is reduced as shown in Fig. 3(b). In Fig. 3(b), the area corresponding second convexity is shown. Figure 3(c) shows the third convexity and its area, Fig. 3(d) shows the area corresponding to fourth convexity and finally Fig. 3(e) shows area corresponding to last convexity. It may be noted that the convexities in Fig. 3(c,d,e) are derived convexities.

3.3 Reduction Rules

The reduction rules are applied only when two consecutive type 1 vertices, i.e., convex edge is detected. Let v_1, v_2, v_3, and v_4 be the four most recent vertices in order and type of v_2 and v_3 be 1. Let v_0 be the vertex (if any) that precedes v_1. If v_0 exists then reduction rule is applied on the sequence $< v_0, v_1, v_2, v_3, v_4 >$ of vertices otherwise it is applied on the sequence $< v_1, v_2, v_3, v_4 >$ of vertices. Depending on the types of vertices v_1 and v_4, there will be four possibilities—(i) 3113, (ii) 3111, (iii)1111 and finally (iv) 1113. Two rules with their sub-rules are formulated, Rule 1 takes care of the pattern in (i) and (iv) and Rule 2 deals

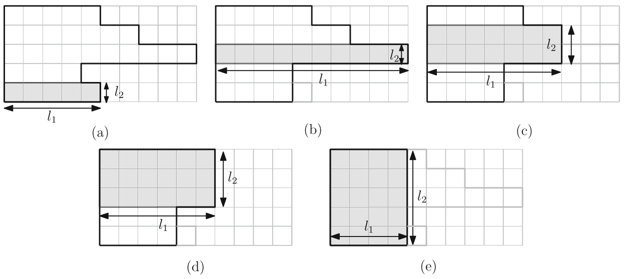


Fig. 3. Illustration of finding largest rectangle in a histogram polygon.

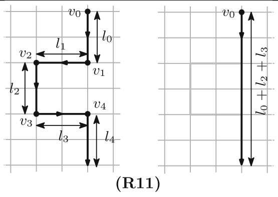
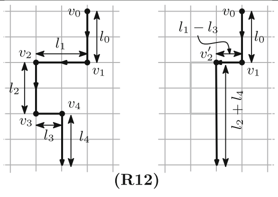
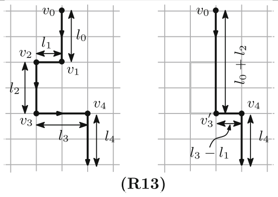
	**Rule R11**$(l_1 = l_3)$: $< v_0(t_0, l_0), v_1(1, l_1), v_2(3, l_2),$ $v_3(3, l_3), v_4(1, l_4) >$ if v_0 exists $\rightarrow < v_0(t_0, l_0 + l_2 + l_4) >$ if v_0 does not exist $\rightarrow < v_1(1, l_2 + l_4) >$
	**Rule R12**$(l_1 > l_3)$: $< v_0(t_0, l_0), v_1(3, l_1), v_2(1, l_2),$ $v_3(1, l_3), v_4(3, l_4) >$ if v_0 exists $\rightarrow < v_0(t_0, l_0), v_1(3, l_1 - l_3),$ $v'_2(1, l_2 + l_4) >$ if v_0 does not exist $\rightarrow < v_1(1, l_1 - l_3), v'_2(1, l_2 + l_4) >$
	**Rule R13**$(l_1 < l_3)$: $< v_0(t_0, l_0), v_1(3, l_1), v_2(1, l_2),$ $v_3(1, l_3), v_4(3, l_4) >$ if v_0 exists $\rightarrow < v_0(t_0, l_0 + l_2), v'_3(1, l_3 - l_1),$ $v_4(3, l_4) >$ if v_0 does not exist $\rightarrow < v_1(1, l_2), v'_3(1, l_3 - l_1), v_4(3, l_4) >$

Fig. 4. Illustration of reduction Rule 1

with the pattern in (ii) and (iii). The proposed algorithm applies reduction rules only when patterns 3113 or 3111 are encountered. The reduction processes are explained below.

Pattern 3113: This pattern implies that two convex (Type 1) vertices preceded concave (Type 3) and followed by another concave vertex. Two consecutive Type 1 vertices in the middle of the pattern creates a convex edge which is to be removed. The reduction process is explained with the help of Fig. 4. Let l_i denote the length of outgoing edge from vertex v_i. Depending on the length l_1 and l_3 there are three sub-rules.

Rule 11 $(l_1 = l_3)$

To remove the convexity, there are two cases to be considered depending on the existence of the vertex v_0. If v_0 exists, vertices v_1, v_2, v_3, and v_4 are removed and the length l_0 is updated to $l_0 + l_3 + l_4$. On the other hand, if v_0 does not exist, vertices v_2, v_3, and v_4 are removed and length l_1 is updated to $l_3 + l_4$.

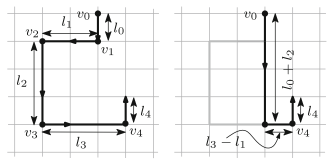
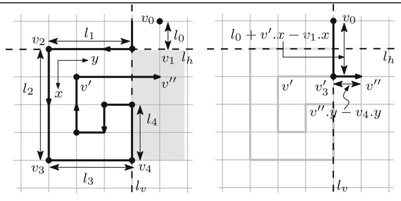
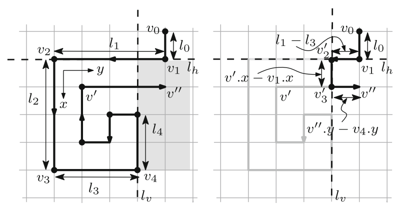
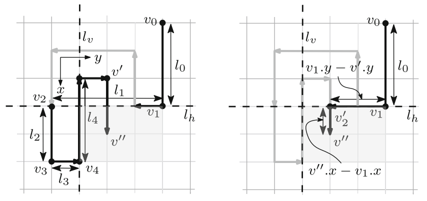
	**Rule R21**$(l_1 < l_3)$: $< v_0(t_0, l_0), v_1(3, l_1), v_2(1, l_2), v_3(1, l_3), v_4(1, l_4) >$ if v_0 exists $\rightarrow< v_0(t_0, l_0 + l_2), v_3'(1, l_3 - l_1), v_4(1, l_4) >$ if v_0 does not exist $\rightarrow< v_1(1, l_2), v_3'(1, l_3 - l_1), v_4(1, l_4) >$
	**Rule R22A**$(l_1 = l_3)$: $(v''.x > v_1.x \;\&\; v''.y > v_4.y \;\&\; v'.x > v_1.x)$ $< v_0(t_0, l_0), v_1(3, l_1), v_2(1, l_2), v_3(1, l_3), v_4(1, l_4) >$ if v_0 exists $\rightarrow< v_0(t_0, l_0 + v'.x - v_1.x), v_3'(1, v''.y - v_4.y) >$ if v_0 does not exist $\rightarrow< v_1(1, v'.x - v_1.x), v_3'(1, v''.y - v_4.y) >$
	**Rule R22B**$(l_1 > l_3)$: $(v''.x > v_1.x \;\&\; v''.y > v_4.y \;\&\; v'.x > v_1.x)$ $< v_0(t_0, l_0), v_1(3, l_1), v_2(1, l_2), v_3(1, l_3), v_4(1, l_4) >$ if v_0 exists $\rightarrow< v_0(t_0, l_0), v_1(1, l_3 - l_1), v_2'(1, v'.x - v_1.x), v_3'(1, v''.y - v_4.y) >$ if v_0 does not exist $\rightarrow< v_1(1, l_3 - l_1), v_2'(1, v_1.x - v'.x), v_3'(1, v''.y - v_4.y) >$
	**Rule R22C**$(l_1 \geq l_3)$: $(v''.x > v_1.x \;\&\; v''.y > v_4.y \;\&\; v'.x \leq v_1.x)$ $< v_0(t_0, l_0), v_1(3, l_1), v_2(1, l_2), v_3(1, l_3), v_4(1, l_4) >$ if v_0 exists $\rightarrow< v_0(t_0, l_0), v_1(1, v_1.y - v'.y), v_2'(1, v''.x - v_1.x) >$ if v_0 does not exist $\rightarrow< v_1(1, v_1.y - v'.y, v_2'(1, v''.x - v_1.x) >$

Fig. 5. Illustration of reduction Rule 2

Rule 12 ($l_1 > l_3$)
To remove the convexity, vertex v_2 is modified to v_2' and its length is set to l_2+l_4. The length l_1 is modified as $l_1 - l_3$ and the vertices v_3 and v_4 are removed. This reduction is independent of presence of the vertex v_0.
Rule 13 ($l_1 < l_3$)
This rule depends on the presence of vertex v_0. If v_0 is present, its length is updated to $l_0 + l_2$, vertices v_1 and v_2 are removed, and vertex v_3 is modified to v_3' with its length set to $l_3 - l_1$. On the other hand, if v_0 is not present, length l_1 is modified as l_2, vertex v_2 is removed, and vertex v_3 is modified to v_3' with its length set to $l_3 - l_1$.

Pattern 3111: This pattern indicates that three consecutive convex (Type 1) vertices is preceded by a concave (Type 3) vertex and may create a convoluted sequence of vertices on the boundary of the object. To remove the convexity of this pattern, the traversal is continued until it comes out of the convoluted region (i.e., the shaded region) bounded by the horizontal line l_h through v_1 and the vertical line l_v through v_4 as shown in Fig. 5.
Rule 21($l_1 < l_3$)
This rule is same as **R13**. If v_0 exists, its length is updated to $l_0 + l_3$, vertices v_1 and v_2 are removed, and vertex v_3 is modified to v_3' with its length modified to $l_3 - l_1$. If v_0 does not exist, length l_1 is modified as l_2, vertex v_2 is removed, and finally vertex v_3 is modified as v_3' with its length modified to $l_3 - l_1$.
Rule 22($l_1 \geq l_3$)
To remove this type of convexity, the traversal is continued from vertex v_4 until it comes out of the convoluted region. This can be checked by comparing the coordinates of the vertices during the traversal. The middle and bottom rows of Fig. 5 illustrate this situation for one direction (downward or direction 3) of vertex v_2. In this case the traversal comes out of the convoluted region when it reaches the first vertex whose x-coordinate value is less than the x-coordinate value of v_4 and the y value is greater than the y value of vertex v_1. As shown in the figure, when the traversal reaches vertex v'', the above condition is satisfied and reduction is applied as follows. Let v' is the immediate previous vertex of v''. If $l_1 = l_3$, length of v_0 if exists, is updated to $l_0 + v'.x - v_1.x$, v_3 is modified to v_3' with its length modified to $v''.y - v_4.y$, and finally vertices v_1, v_2 and all the the vertices from v_4(including it) to v' are deleted. If v_0 does not exist, length of v_1 is updated to $v'.x - v_1.x$, vertex v_3 is modified to v_3' with its length modified to $v''.y - v_4.y$, and finally vertices v_2 and all the the vertices from v_4(including it) to v' are removed. This is explained by rule **R22A** in Fig. 5. If $l_1 > l_3$, length of v_1 is updated to $l_1 - l_3$, v_2 is modified to v_2' with its length modified to $v'.x - v_1.x$, v_3 is modified to v_3' with its length modified to $v''.y - v_4.y$, and finally all the vertices from v_4(including it) to v' are removed. In this case the reduction process is independent of existence of v_0. This is explained by rule **R22B** in Fig. 5. It is to be noted that after reduction rule is applied there still exists a convexity formed by vertices v_1, v_2', v_3' and v'' which will be removed subsequently by the application of one of the available rules.

3.4 Algorithm

The Algorithm FIND-RECT (Algorithm 1) is used to determine a largest rectangle in a digital object, which takes as input the digital object, A, and the largest rectangle is reported as the output. Inner isothetic cover of A is generated by calling the procedure IIC (Step 1) and L, L_x, and L_y are created (procedure IIC is based on Sect. 2 [2,3]). L_{curr} determines the current position of the vertex L whereas L_{end} is the last vertex in the vertex list L. Initially, L_{curr} is set to the start of L, i.e., L_{start} and the area of global largest rectangle, *rect.area*, is set to '0' (Step 2). The list L is traversed until it reaches the end L_{end} (Steps 3-6). The procedure CHECK-CONVEX checks whether there is a convex edge (Step 4) and if it is so, procedure FIND-HIST is called (Step 5). L_{curr} advances one step in L (Step 6). In the Procedure FIND-HIST (Procedure 1), L_H stores all the vertices of histogram polygon in anticlockwise manner. Initially L_H is set to NULL (Step 1). SEARCH-NEXT procedure finds out the next vertex, v, of the histogram polygon (explained in Sect. 3.1) (Step 2). The vertices v_1, v_2, and v are appended in L_H (Step 3). All the vertices in histogram polygon has to be found out till it is in range (Steps 4-6) which is determined by the procedure CHK-RANGE (Step 4). In Step 5, the next vertex, v' in histogram polygon is found out by the procedure SEARCH-NEXT (Step 5) and v' is appended in L_H (Step 6). The FIND-RECT (Procedure 2) procedure is called to determine the rectangles in histogram polygon (Step 7). Reduction rules (discussed in Sect. 3.3) are applied on convex edge from which histogram polygon has been generated, by calling the procedure APPLY-RULE (Step 8). The L_{curr} will be updated accordingly (Step 9) and will be returned to FIND-LR (Step 10).

In the Procedure 2, FIND-RECT, L_{H_v} is initialized to the next vertex of the convex edge in anticlockwise manner in the histogram polygon (Step 1). Steps 3–9 are repeated until the condition in Step 2 is false, i.e., all the vertices of the histogram polygon will be traversed except its base. If a convex edge is detected by calling the procedure CHECK-CONVEX in Step 3, corresponding rectangle, *rect'*, is determined by calling the procedure CAL-RECT (Step 4). Corresponding area of the rectangle is determined by the procedure CAL-AREA (Step 5). The area of *rect'* is compared with *rect*, if the area of *rect'* is larger, then the global largest rectangle is updated (Steps 6-7). Reduction rules (discussed in Sect. 3.3) are applied on the convex edge (Step 8). L_{H_v} advances one step in L (Step 9).

Algorithm 1. FIND-LR

```
Input: $A$
Output: $rect$
1 $L, L_x, L_y \leftarrow$ IIC(A) ;
2 $L_{curr} \leftarrow L_{start}, rect.area \leftarrow 0$;
3 while $L_{curr} \neq L_{end}$ do
4     if CHECK-CONVEX($L_{curr} \rightarrow type, (L_{curr} \rightarrow next) \rightarrow type$) then
5         FIND-HIST($L_{curr}, L_{curr} \rightarrow next$)
6     $L_{curr} \leftarrow L_{curr} \rightarrow next$;
7 return $rect$;
```

Procedure 1. Find-Hist(v_1, v_2)

```
1 $L_H \leftarrow \{\phi\}$;
2 $v \leftarrow$ SEARCH-NEXT($v_2, L_x, L_y$);
3 $L_H \leftarrow L_H \cup \{v_1, v_2, v\}$;
4 while CHK-RANGE($v_1, v_2$) do
5     $v' \leftarrow$SEARCH-NEXT($v, L_x, L_y$);
6     $L_H \leftarrow L_H \cup \{v'\}$;
7 FIND-RECT($L_H$);
8 APPLY-RULE($v_1, v_2$);
```

Procedure 2. Find-Rect(L_H)

```
1 $L_{H_v} \leftarrow L_{H_{start}} \rightarrow next \rightarrow next$;
2 while $L_{H_v} \neq L_{H_{end}}$ do
3     if CHECK-CONVEX( $L_{H_v} \rightarrow type$, ($L_{H_v} \rightarrow next$) $\rightarrow type$) then
4         $rect' \leftarrow$ CAL-RECT($L_{H_v}, L_{H_v} \rightarrow next$);
5         $rect'.area \leftarrow$ CAL-AREA($rect'$) ;
6         if $rect'.area > rect.area$ then
7             $rect \leftarrow rect'$;
8         APPLY-RULE($L_v, L_v \rightarrow next$);
9     $L_{H_v} \leftarrow L_{H_v} \rightarrow next$;
```

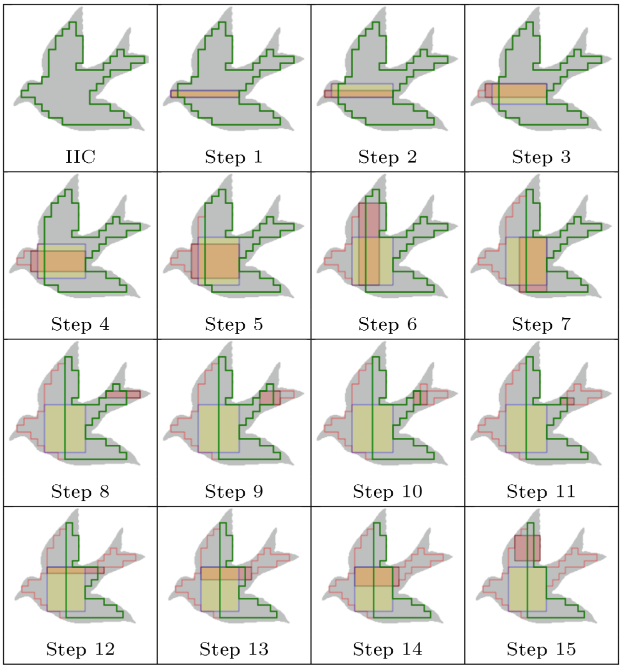


Fig. 6. Demonstration of the proposed algorithm on an object (Bird)

3.5 Demonstration

An illustration of obtaining largest rectangle is shown in Fig. 6. The top-left figure shows the IIC of the object, the green polygons in each step indicates the reduced polygon, the yellow rectangle indicates the largest rectangle found so far and the pink polygon indicates the largest rectangle for immediate previous convexity or the current convexity. Step 1 shows the result of application of reduction rule **R13** for the first convexity it encounters and the largest rectangle (in yellow) found corresponding to this convexity. Step 2 shows the removal of second convexity with rule **R12**. Since the largest area rectangle obtained for this convexity is greater than the largest rectangle obtained so far (before this), the global largest rectangle is updated with this rectangle. Continuing this way the convexity at Step 5, gives largest rectangle for this object since the rectangles corresponding to all subsequent convexities are smaller. It is to be noted that reduction rules are applied for patterns 3113 and 3111, so the IIC will be reduced to a histogram polygon whose base will be the bottom edge of the reduced polygon. At the last step the largest rectangle for this histogram polygon is to be computed for potential largest rectangle. In this case the largest rectangle for this reduced polygon is smaller than the one already computed.

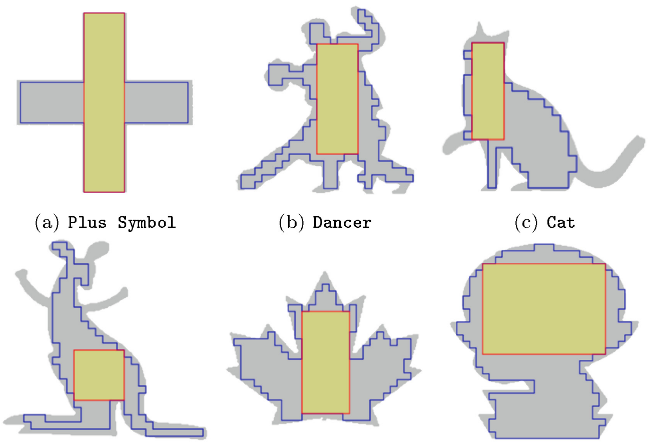
3.6 Time Complexity

To estimate the running time of the algorithm, let us look at the steps involved and their individual running time. Also, let n be the number of pixels on the contour of the digital object, A and g be the grid size. To construct inner isothetic cover along with L, L_x, and L_y $O((n/g)\log(n/g))$ time is required. To determine the largest rectangle, the inner isothetic cover is linearly traversed once and whenever an unexplored convex edge is encountered the procedure Find-Hist and thereby procedure Find-Rect is called to find the largest rectangle corresponding to this convex edge. To find the histogram polygon with respect to a convex edge, $O(n/g + \log(n/g))$ time is required, as it includes searching in L_x and L_y which takes $O(\log n/g)$ time. Procedure Check-Convex, Chk-Range, Apply-Rule, Update, Cal-Area, and Find-Rect take $O(1)$ time. Procedure Find-Rect traverses linearly only the vertices lying opposite to the base of the histogram polygon found out by the procedure Find-Hist. So, Find-Rect takes $O(n/g)$ time to calculate the largest rectangle inside the histogram polygon. It is to be noted that each convex edge is traversed only once and the other convex edges which are in opposite direction but totally contained within the convex edge currently being considered need not be checked as largest rectangle corresponding to these convex has already been taken care of. If there are in total k number of convex regions where $k \ll n$, then total time complexity will be $O(k.n/g + (n/g)\log(n/g))$.

3.7 Proof of Correctness

The algorithm identifies a convex edge and finds out the corresponding histogram polygon, then the largest rectangle inside the histogram polygon is computed. After a convex edge is considered, it is reduced following a certain combinatorial reduction rules. After the reduction if it gives rise to secondary convex, it is treated in the similar manner stated above to find the corresponding largest rectangle. The largest of all these rectangles corresponding to each convex edge is reported as largest rectangle. To prove that the algorithm indeed finds out largest rectangle we have to show that the algorithm considers all convex edges.

The traversal procedure ensures that all convex edges (and secondary convex edges) are considered as it starts from the top left vertex of the IIC and continues till the traversal reaches the start point. Sides of a largest rectangle, as it tries to maximise the area, will either be a convex edge or a concave edge may constitute a part of its side. The procedure for finding out largest rectangle inside a histogram polygon of a convex edge, ensures that it finds a largest rectangle as it walks along the sky-line of the histogram polygon (reduces whenever required) and determines the rectangle it produces with the base (the convex edge). To prove that the algorithm also terminates, it is to be noted that during the traversal, whether or not a convex edge is encountered, the traversal advances to the next vertex of IIC. So, eventually the traversal reaches the start point and terminates.



(a) `Plus Symbol` (b) `Dancer` (c) `Cat`

(d) `Kangaroo` (e) `Mapple Leaf` (f) `Cartoon`

Fig. 7. Largest area rectangle (shaded in yellow) inside six different objects for $g = 8$.

Table 1. Different data for various digital objects at grid size, $g = 8$.

Object	Object perimeter	Object area	Perimeter of IIC	Area of IIC	Perimeter of LR	Area of LR
`Bird`	625	27768	688	7168	208	2688
`Kangaroo`	1151	57828	1088	11008	224	3136
`Dancer`	1057	50398	1320	14400	352	6144
`Cartoon`	918	65536	1152	35456	528	17024
`Cat`	730	42920	616	8896	256	2816
`Hand`	1300	72675	1632	28224	464	13312
`Mapple leaf`	861	50882	992	17472	352	6720
`Plus Symbol`	978	70488	960	23744	608	13888

4 Experimental Results

The proposed algorithm is implemented in C in Ubuntu 12.04, 64-bit, kernel version 3.5.0-43-generic, the processor being Intel i5-3570, 3.4 GHz FSB. Experimental results of six different digital objects are shown in Fig. 7. In Table 1, different data, e.g., area and perimeter of digital object, IIC, and largest rectangle, are given. It is seen that largest rectangle occupies approximately one-third area of inner isothetic polygon. The results show that the algorithm can be useful for shape analysis of digital object, as most of the time the largest rectangle determined is at the central position of the digital object. A digital object my be characterized by positioning the successive largest rectangle inside it. These type of information may be useful to determine some topological information from digital objects when the largest rectangles are placed recursively in the rest of the IIC.

5 Conclusion

This paper describes a combinatorial algorithm to find largest rectangle inside a digital object in $O(k.n/g+(n/g)\log(n/g))$ time. The paper also presents the rules for the algorithm, a demonstration, and time complexity. Experimental results show the efficacy of the algorithm. The algorithm can be applied for shape analysis of digital object. LR problem has some industrial application also which has been stated in Sect. 1. One largest rectangle can divide the object in several parts. We can generate iteratively largest rectangle in each part upto a certain limit. LR in each level will form a tree which will represents the connectivity among several parts inside the object. In future, some topological information for the digital objects can be derived from above mentioned technique. All these implies the practical importance of the problem in various shape related applications.

References

1. Aggarwal, A., Suri, S.: Fast algorithms for computing the largest empty rectangle. In: Proceedings of the 3rd Annual Symposium on Computational Geometry, pp. 278–290 (1987)
2. Biswas, A., Bhowmick, P., Bhattacharya, B.B.: TIPS: on finding a tight isothetic polygonal shape covering a 2D object. In: Kalviainen, H., Parkkinen, J., Kaarna, A. (eds.) SCIA 2005. LNCS, vol. 3540, pp. 930–939. Springer, Heidelberg (2005)
3. Biswas, A., Bhowmick, P., Bhattacharya, B.B.: Construction of isothetic covers of a digital object: A combinatorial approach. J. Vis. Comun. Image Represent. **21**(4), 295–310 (2010)
4. Chang, J., Yap, C.: A polynomial solution for the potato-peeling problem. Discrete Comput. Geom. **1**, 155–182 (1986)
5. Chazelle, B., Drysdale, R.L., Lee, D.T.: Computing the largest empty rectangle. In: STACS-1984, pp. 43–54. Springer, Heidelberg (1984)
6. Chazelle, B., III, R.D., Lee, D.: Computing the largest empty rectangle. SIAM J. Comput. **15**, 300–315 (1986)

7. Daniels, K., Milenkovic, V., Roth, D.: Finding the largest area axis-parallel rectangle in a polygon. Comput. Geom.: Theor. Appl. **7**, 125–148 (1997)
8. Klette, R., Rosenfeld, A.: Digital Geometry: Geometric Methods for Digital Picture Analysis. Morgan Kaufmann, San Francisco (2004)
9. McKenna, M., O'Rourke, J., Suri, S.: Finding the largest rectangle in an orthogonal polygon. In: Proceedings of the 23rd Allerton Conference on Communication, Control and Computing, pp. 486–495 (1985)
10. Nandy, S.C., Bhattacharya, B.B., Ray, S.: Efficient algorithms for identifying all maximal isothetic empty rectangles in VLSI layout design. In: Nori, K.V., Veni Madhavan, C.E. (eds.) Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 472, pp. 255–269. Springer, Heidelberg (1990)
11. Ullman, J.: Chap. 9: Algorithms for VLSI Design Tools. Computational Aspects of VLSI. Computer Science Press, Rockville (1984)