International Journal of Scientific & Engineering Research, Volume 6, Issue 5, May-2015
ISSN 2229-5518

Advancement in Guard Zone Computation through Detection and Exclusion of the Overlapped Regions

Ranjan Mehera, Piyali Datta, Arpan Chakraborty, and Rajat Kumar Pal

Abstract— The guard zone computation problem claims utmost importance in VLSI layout design, where the circuit components (or the functional units/modules or groups/blocks of different sub-circuits) that may be viewed as a set of polygonal regions on a two-dimensional plane, are not supposed to be placed much closer to each other in order to avoid electrical (parasitic) effects among them. Each (group of) circuit component(s) C_i is associated with a parameter δ_i such that a minimum clearance zone of width δ_i is to be maintained around C_i. Beyond this, it has huge significance in the field of robotic motion planning, geographical information system, automatic monitoring of metal cutting tools, and design of embedded systems. If the guard zonal regions overlap, we have to remove the overlapped regions in order to compute the resultant outer guard zone (sometimes inner guard zones are also an issue to be considered). In this paper, we have developed an algorithm to compute the guard zone of a simple polygon as well as to exclude the overlapped regions among the guard zonal segments (if any) in O(n log n) time, where n is the number of vertices of the given simple polygon.

Index Terms— Simple polygon, Safety zone, Notch, Convex hull, False hull edge, Convolution, Minkowski sum.

1 Introduction

2D guard zone computation problem undoubtedly occupies a majority of interest in the field of VLSI physical design automation and design of embedded systems, as resizing is an important problem in VLSI layout design as well as in embedded system design. If two or more subcircuits are close enough, it may result in parasitic effect which disrupts the performance of the circuit. Thus, with respect to resizing problems in VLSI, this is the motivation of defining the safety zone of a polygon [6]. The width of the safety zone depends on the polygon, i.e. the type of the circuit present therein and the free space required to perform all the tasks safely. In this context, it is also essential to detect overlapping (if any) among the guard zonal regions and accordingly remove them to obtain resultant guard zone. Through this procedure we may also find a hole (if any) in the guard zone and some times those holes may be used to place a subcircuit in order to utilize the chip area more densely and efficiently. In this paper we develop an algorithm to compute guard zone of a 2D simple polygon including the detection and exclusion of the overlapped regions (if any).

To solve the guard zone computation problem, one of the reputed methods is Minkowski sum [3]; that can be used to draw a parallel line with respect to a given polygonal edge. Essentially, Minkowski sum between a line and a point with same x- and y-coordinates gives a line parallel to the given one. But the question arises is whether the parallel line is inside or outside the polygon. Here the definition of Minkowski sum [3] can be extended as below.

If A and B are subsets of R_n, and $\lambda \in R$, then $A+B = \{x+y \mid x \in A, y \in B\}$, $A-B = \{x-y \mid x \in A, y \in B\}$, and $\lambda A \{\lambda x \mid x \in A\}$.

Note that $A+A$ does not equal $2A$, and $A-A$ does not equal 'zero' in any sense.

Another method may be used to compute guard zone of a polygon. The convolution [1] is used to solve the guard zone problem. The convolution between a polygon and a circle of radius r gives us the desired solution. But the circles need to be drawn in every possible points of the polygon and consequently the time complexity of the algorithm increases. Minkowski sum and convolution theories find their vast applications in Mathematics, Computational geometry, resizing of VLSI circuit components and/or recently in embedded systems, and in many problems in many other subjects.

2 Literature Survey

In the context of guard zone computation, several different algorithms have been proposed so far. The most discussed tool for guard zone computation is the Minkowski Sum [4]. Apart from Minkowski sum, convolution can also be used as a tool for guard zone computation. Essentially, Minkowski sum between a line (as polygonal segment) and a point (perpendicularly at a distance r apart) with the same x- and y-coordinates gives a line parallel to the given one. But the question arises is whether the parallel line is inside or outside the polygon. We have already defined Minkowski sum in the introduction of this paper.

The convolution between a polygon and a circle of radius r may give us the desired solution of a guard zone. But the circles need to be drawn in every possible point of the polygon and thus the time complexity of the algorithm increases. The

Ranjan Mehera, Piyali Datta, Arpan Chakraborty, and Rajat Kumar Pal are from Department of Computer Science and Engineering, University of Calcutta

Acharya Prafulla Chandra Roy Siksha Prangan, JD – 2, Sector – III, Saltlake City, Kolkata – 700 098, West Bengal, India

ranjan.mehera@gmail.com, piyalidatta150888@gmail.com, arpanc250506@gmail.com, pal.rajatk@gmail.com

International Journal of Scientific & Engineering Research Volume 6, Issue 5, May-2015
ISSN 2229-5518

complexity of computing Minkowski sum of two arbitrary simple polygons P and Q is $O(m^2n^2)$ [4], where m and n are the number of vertices of these two polygons. In particular, if one of the two polygons is convex, the complexity of Minkowski sum reduces to $O(mn)$. In [6], numerous results are proposed on the Minkowski sum problem when one of the polygons is monotone.

In this context, a linear time algorithm is devised for finding the boundary of minimum area guard zone of an arbitrarily shaped simple polygon in [9]. This algorithm uses the idea of Chazelle's linear time triangulation algorithm and requires space complexity of $O(n)$ as well [2], where n is the number of vertices of the polygon. After having the triangulation step, this algorithm uses only dynamic linear and binary tree data structures. Again, a time-optimal sequential algorithm for computing a boundary of guard zone has been developed in [5-7] that uses simple analytical and coordinate geometric concepts. The algorithm can easily be modified to compute the regions of width r outside the polygon as guard zone, and also inside the polygon.

3 Problem Formulation

In this section, we formulate and develop a comprehensive algorithm to compute guard zone of a simple polygon while detecting and excluding the overlapped regions (if any) of the guard zone. A simple polygon may contain both convex and concave vertices in it. We define these vertices as follows: A vertex v of a polygon P is defined as convex (concave), if the angle between its associated edges inside the polygon, i.e. the internal angle at vertex v, is less than or equal to (greater than) 180^o. In Fig. 1, angles at vertices v_4, v_5, and v_6 are concave whereas angles at vertices v_1, v_2, v_3, v_7, v_8, and v_9 are convex. To know whether an angle θ, inside the polygon, is either convex or concave at vertex v, we do a constant time computation of determining the value (of θ) at vertex v. For a given set of three or more connected vertices that form a simple polygon, the orientation of the resulting polygon is directly related to the sign of the angle at any vertex of the convex hull of the polygon. For example, to determine the type of angle formed between edges $a = (X_A, X_B)$ and $b = (X_B, X_C)$ with coordinates X_A (x_1, y_1), X_B (x_2, y_2), and X_C (x_3, y_3), the following equation is being used that takes constant time for finding out the angle whether it is convex or concave:

$$\det(O) = (X_B - X_A)(Y_C - Y_A) - (X_C - X_A)(Y_B - Y_A).$$

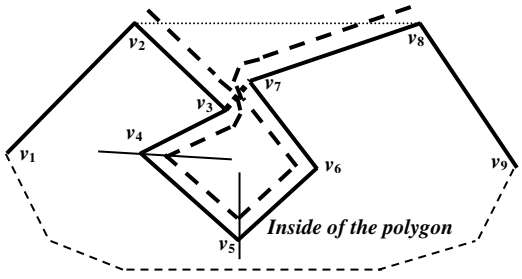

Fig. 1: A notch is formed inside (or below) the false hull edge formed by vertices v_2 and v_8, and a guard zone is obtained for this notch as shown by dotted lines and circular arcs outside of the polygon.

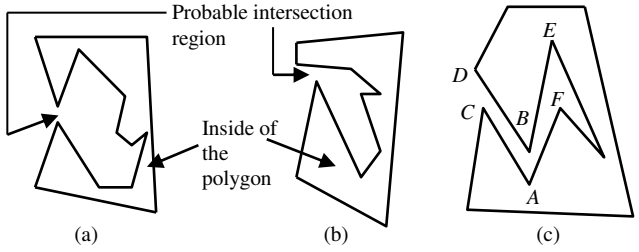

Fig. 2: (a) Different probable overlapping of guard zone of two convex regions. (b) Probable overlapping of guard zone of one convex region with a straight line segment. (c) A polygon consisting of a number of convex (e.g., *B*, *C*, *D*, and *F*) and concave (e.g., *A*, *E*, and *G*) vertices whose guard zonal regions are probable to overlap.

Now, if all the n external angles of the polygon are convex, the guard zone is computed only with the help of n straight line segments and n circular arcs in linear time [6, 7]. Problems may arise in computing guard zone for the portions of polygon P with concave external angles. In this context, we introduce the concept of notch. A *notch* is a polygonal region outside a polygon that is starting and terminating between two consecutive convex vertices of the polygon that are not adjacent. As our objective is to compute the intersection points and exclude the overlapped regions (if any), we first compute the guard zone G (without excluding overlapped portions) of the given simple polygon P in linear time [6, 7]. The difficulty arises while excluding the part(s) of G that overlap(s).

As a guard zone consists of only line segments and circular arcs, intersection may occur between two line segments, one line segment and a circular arc or two circular arcs. In Figs. 2(a), 2(b) and 2(c), different kinds of probable overlapping sections have been depicted. To find out all the intersection points and exclude the overlapped region(s) (in order to get the desired guard zone only, including holes, if any, as parts of G), we may execute an $O(n^2)$ algorithm for each pair of such segments, among all straight line segments and circular arcs resulting and ensuring inefficiency with respect to cost. In the next section, we present a number of algorithms which solves the guard zone computation problem bounding its cost in time to $O(n \log n)$.

4 Algorithm Development

To detect and exclude the overlapped guard zonal regions we use the line sweep algorithm among the guard zonal segments that in turn may reduce the overall complexity of guard zone computation algorithm to $O(n \log n)$, instead of $O(n^2)$ [6]. As the guard zone is a set of line segments and circular arcs and line sweep algorithm can only be applied on a set of line segments, we cannot solve our problem directly through line sweep algorithm. Hence, our incline to modify the guard zone in such a way that we may use line sweep algorithm, which is an efficient tool for intersection checking.

The computation procedure will be trickier, if we somehow detect the probable intersection region or the components which are most prone to overlap and afterwards we detect the final intersecting guard zonal component pair(s). Also it may

International Journal of Scientific & Engineering Research Volume 6, Issue 5, May-2015
ISSN 2229-5518

happen that originally there were no intersections but the algorithm reports the intersection points and works only with those intersection information and finally reports the computed guard zone. The algorithm works in two phases. In Phase I, it detects the prone guard zonal components to be intersected and in Phase II, it computes the guard zone eliminating overlapped portions, if any.

The main objective of this algorithm is to reduce the search space so that there will be a speed up in the application of the line sweep algorithm. We notice that for each of the circular arcs, its two tangents, which are actually the line segments of the guard zone attached to that circular arc, can be extended. Thus, they meet at a point. If we apply this at each such circular arc we get s number of points for s number of circular arcs. As a result the guard zone will become a polygon which is not necessarily a simple one. We call it the extended or overestimated guard zone. By converting a guard zone into an extended or overestimated guard zone we now do not have to consider the circular arc portions for repetitive bisection.

For an example, the vertex p of the overestimated polygon as shown in Fig. 3, which is formed by extending two neighbouring line segments of the circular guard zonal region of the convex vertex v of the original polygon. Here the circular arc $v'v''$ is the guard zonal region of the convex vertex v. Now the neighbours of this circular arc are $u'v'$ and $v''w'$. They are extended and meet at p, which is considered to be a convex vertex of the overestimated polygon. Thus, all the circular arcs of the guard zone are now replaced by corresponding convex vertices in the extended polygon. Now if the extended guard zone, which is actually a polygon, becomes also a simple polygon it means that there is no intersection among the guard zonal components and there is no need of Phase II of the algorithm.

Here the line sweep algorithm is applied where the input is a set of line segments (original guard zonal line segments and the derived line segments replacing the circular arcs) associated with their starting and ending coordinates as event points. The starting point, ending point, and point of intersection (if any) are the three types of event points. As usual all the event points are sorted and the sweep line is traversed through the sorted list of event points. At each event point, insertion (at start event point), deletion (at end event point), and update (at intersection point) of neighbouring operations are performed during the sweep line traversal. Here the data structures used are of the highest importance to accomplish the task in O(n log n) time.

This algorithm maintains two binary search trees, one for storing the event points and another for the line segments to keep track of their neighbouring information. All the three operations are performed on the query tree. As an intersection occurs only between the neighbours, this data structure reduces the search space for intersection checking to the set of neighbours of a line segment. Furthermore, it is proved that line sweep algorithm reports all the intersection points. After getting all these intersection points, we traverse the original guard zone and depending on the intersection points we exclude the intersected or overlapped region(s), and report the outer and inner guard zone [5], accordingly.

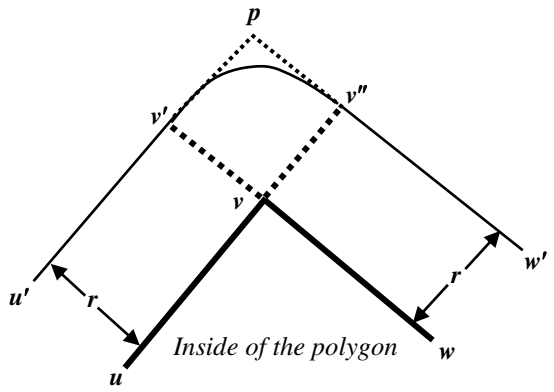

Fig. 3: Extending the guard zonal line segments ($u'v'$ and $w'v''$) associated with a circular arc ($v'v''$) to obtain the overestimated guard zonal region that meets at p.

When applying the line sweep algorithm to a polygon, we need to remember that each pair of consecutive edges share one event point (starting or ending) and thus the algorithm needs a little bit modification as it takes the set of line segments those do not share their start or end point. At the beginning of the algorithm, the sweep line is at the maximum or minimum event point; again that point is the starting point of two line segments and hence arises the ambiguity in selecting the root of the query tree as both the line segments are claimers to be the root. We remove this confusion in the following way; let us consider $L1$ and $L2$ are such line segments competing to be the root.

First we compare the end points of the line segments. If their y-coordinate values are different, the line segment having larger y-coordinate value (say $L1$) is selected as the root and another line segment (i.e. $L2$) has been selected as its left (right) child if the x-coordinate value of $L2$ is less (greater) than that of $L1$. On the other hand, if their y-coordinate values are same, the line segment having smaller y-coordinate value (say $L1$) is selected as the root and another line segment (i.e. $L2$) has been selected as its right child.

Another modification of line sweep algorithm lies in the fact that we have to deal with a closed polygon and the query tree is empty only at the starting and at the termination of the algorithm, whereas in the traditional line sweep algorithm to have an empty query tree is not a termination condition, because it may happen that at the end point of a line segment where there is no other line segment whose start point is above that very point and the end point is below it. Except these two above cited modifications, the algorithm proceeds in the way of traditional line sweep algorithm identifying all the intersection points, if exists.

Now in Phase II, the algorithm deals with the original guard zonal regions, not with the extended guard zones and with the regions which are detected to be the probable intersection regions by the first phase of the algorithm. As has been discussed earlier, these regions are subdivided into line segments (if there is any circular arc in these parts of the polygon) and the line sweep algorithm is further applied on these line segments only. At the end of the second phase, we get the original intersections and depending on these points,

International Journal of Scientific & Engineering Research Volume 6, Issue 5, May-2015
ISSN 2229-5518

the algorithm reports the outer guard zone as well as the inner guard zonal loop, if any. Two phases of the algorithm will be discussed next with the help of an example.

Let us consider a simple polygon S whose vertices are stored in anticlockwise manner as a, b, c, d, e, f, g, h, i, j, k, l, where a, b, c, d, e, f, g, k are convex and h, i, j are concave vertices (Fig. 4(a)). After obtaining the guard zone of this polygon, the algorithm is applied to detect and exclude the overlapping portions.

4.1 Phase I of the Algorithm

As the pair of neighbouring line segments of each circular arc (guard zone of a convex vertex) is extended, that meet at a point which is again a convex vertex of the overestimated polygon. Now let us consider a overestimated polygon X as shown in Fig. 5(b), whose edges are labeled as AB (2), BC (3), CD (4), DE (5), EF (6), FG (7), GH (8), HI (9), IJ (10), JK (11), KL (12), LA (1).

After sorting all the event points in the descending y-value, we get the event list as L, E, A, F, K, J, D, G, H, I, B, C. At the beginning, this array only contains all the start and end points in their sorted sequence (according to their y-coordinates), but subsequently checking of intersection introduces more event points as the intersection points are also considered as the event points. All the event points are handled by the query tree (T).

Now the sweep line is adjusted, parallel to x-axis at the maximum y-coordinate, i.e. at L. It is the starting point of two line segments 1 and 12. The x-coordinate of the end of 1 is less than the x-coordinate of the end of 12. So, in T, 1 is left neighbour of 12. Accordingly, both the trees are shown in Figs. 5(a) and 5(b) conveying this relationship. Though at a glance it seems to be ambiguous to choose the tree structure between these two, this does not introduce any indefiniteness in our algorithm. We can always choose the root by comparing the y-coordinates of the two line segments whose start points are same. The line segment, whose end point is of greater y-coordinate between the two, is selected as the root. Now the other line segment is inserted into the tree as right child or left child of the root depending on the neighbouring relationship of this segment with the root. Here 1 will be the root node and 12 will be its right child as the y-coordinate of A which is the end point of segment 1, is higher than that of K, i.e. the y-coordinate of A is greater than that of K, which is the end point of segment 12. So the tree of Fig. 5(b) is chosen. Now 1 and 12 are consecutive polygonal edges, there is no need of checking intersection between them. L is deleted from the event list. Hence, the content of the event queue becomes E, A, F, K, J, D, G, H, I, B, C.

Next event point is E. It is the starting point of 6 and 5; hence, these two event points are to be inserted into the query tree. As E is a point with higher x-coordinate value than L, 6 and 5 are inserted in the right subtree of 12. Here we compare the end points of 6 and 5 to decide which the direct neighbour of 12 is. As the end point of 6 possesses lower x-coordinate than that of the end point of 5, 6 is inserted first as the right child of 12 as shown in Fig. 5(c). As we like to make the tree height balanced always, rotations are applied as needed and obtain the tree as shown in Fig. 5(d). Now, checking for intersections between 6 and 12 is performed. It results in an intersection point Q. The point is inserted in the intersection list with information Q (6,12) maintaining its order. Again, the intersection point Q is inserted into the event queue maintaining proper y-value sequence. Now, 5 becomes the right neighbour of 6. Accordingly, the tree is shown in Fig. 5(e). E is deleted from the event queue and the content of event queue becomes A, Q, F, K, J, D, G, H, I, B, C.

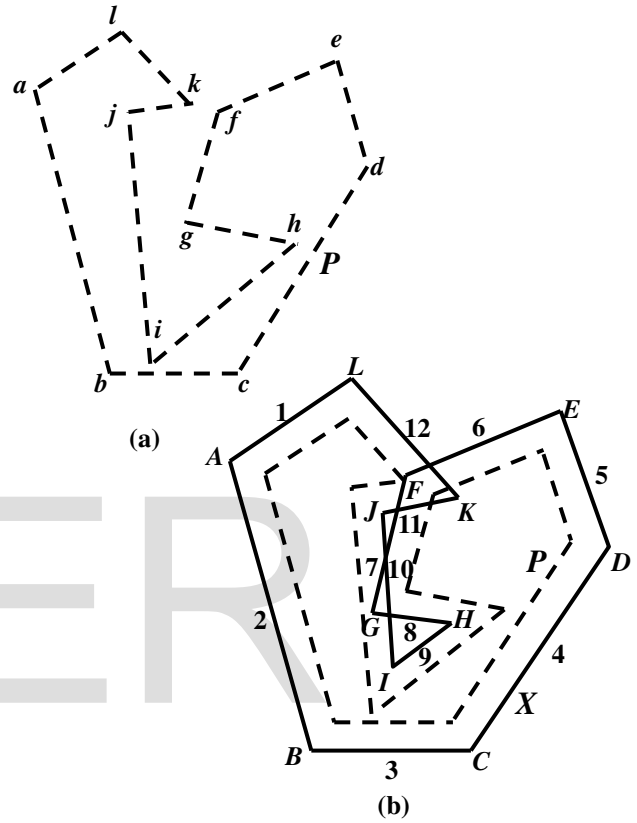

Fig. 4: (a) A simple (dotted) polygon P. (b) The extended guard zone X of the polygon P (X is also a polygon drawn by solid lines).

Now the sweep line moves towards the next event point A, which is the end point of 1 and start point of 2. As 1 is a leaf node in T and the remaining tree does not become height imbalanced even after deleting node A, it is deleted without any additional modification in the tree. As 1 has been deleted and A is of less x-coordinate than the point at which 12 cuts the sweep line, 2 is inserted being the left neighbour of 12. Accordingly, the tree is shown in Fig. 5(f). A is deleted from the event list. The event list is now Q, F, K, J, D, G, H, I, B, C. Now the event point Q is to be handled. As it is an intersection point, the neighbouring information are updated in T. this is performed by interchanging 12 and 6. Again their set of neighbours has also been interchanged; the tree is shown in Fig. 5(g). Q is deleted from the event list. The event list is now F, K, J, D, G, H, I, B, C. The next event pont is F, which is the end point of 6 as well as the start point of 7. Now, 6 is deleted and 7 is inserted at the place of 6 in the query tree. Hence, 7 is the neighbour of 2 and 12 and intersection checking is performed among those. As no intersection is reported the

International Journal of Scientific & Engineering Research Volume 6, Issue 5, May-2015
ISSN 2229-5518

intersection list remains the same and *F* is deleted from the event queue. The resulting query tree structure has been shown in Fig. 5(h). Thus, during the algorithm, the sweep line traverses all the points in event list. In the meantime it handles the query tree *T*. In Fig. 6, all the intermediate trees are shown. As we are applying line sweep algorithm on a bounded region, i.e. a polygon, the query cannot be empty until all the event points are accessed. Hence, the algorithm of Phase I is over when the query tree becomes empty. The status of event queue and the intersection list at each event point have been shown through Table I.

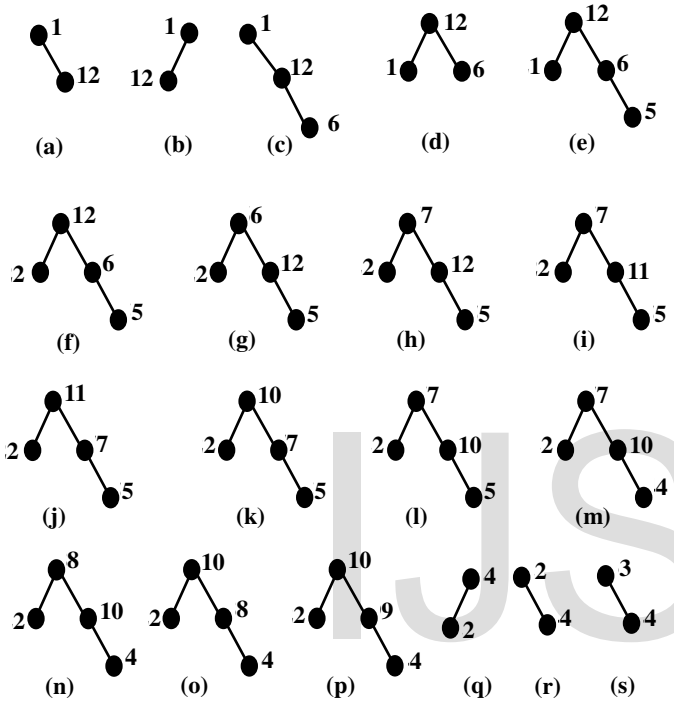

Fig. 5: (a) 1 and 12 have been inserted, 12 being right child of 1. (b) 1 and 12 have been inserted, 12 being left child of 1. (c) 6 has been inserted being right child of 12 and the tree gets imbalanced. (d) The unbalanced tree in (c) is made height balanced. (e) 5 has been inserted as the right child of 6. (f) 1 is deleted and 2 is inserted. (g) The positions of 12 and 6 have been interchanged. (h) 6 is deleted and 7 is inserted. (i) 12 is deleted and 11 is inserted. (j) The positions of 7 and 11 have been interchanged. (j) 11 is deleted and 10 is inserted. (l) The positions of 7 and 10 have been interchanged. (m) 5 is deleted and 4 is inserted. (n) 7 is deleted and 8 is inserted. (o) The positions of 8 and 10 have been interchanged. (p) 8 is deleted and 9 is inserted. (q) 9 and 10 are deleted resulting 4 to be the root. (r) 9 and 10 are deleted (from (p)) resulting 2 to be the root and this tree is selected as the x-coordinate of the end point of 2 is less than that of 4, though both of them have same y-coordinates. (s) 2 is deleted and 3 is inserted at the position of 2.

4.2 Phase II of the Algorithm

At the end of Phase I, we have obtained all the probable intersection points for the overestimated guard zonal regions. Each intersection point can be on the actual guard zonal component or on the extended portion of the guard zonal line segments. So, an intersection point thus obtained may be original or fake depending on whether it is on the original guard zonal segment (or not), and thus, a further checking is required to identify the real ones.

Now, there are three types of intersections: (a) circular arc-circular arc, (b) line segment-circular arc, and (c) line segment-line segment. We denote each intersection point as a triple ⟨intersection point, (two ends of one of the intersecting components), (two ends of the other intersecting component)⟩. Thus, one intersecting pair resulted in the first phase identifies a set of consecutive original guard zonal segments, i.e. ⟨arc-line segment-arc⟩, or ⟨arc-line segment⟩, or ⟨line segment⟩ depending upon the guard zonal segment that has been extended in Phase I. Irrespective of the cases occurred, we have to check a constant number of guard zonal segments whether there is any intersection. Now, during the guard zone computation phase for each of the polygonal edge and vertices, we get the equation of both the line segment and circular arc along with their end points. Hence, the checking between any two of these segments is a constant time operation using simple coordinate geometry. For an example, if we want to check whether a line segment and a circular arc are intersecting, we only need to know the (coordinate) equations of the line segment and the circle. Subsequently, if there is any intersection point, we have to check whether the point is on the circular arc or on the remaining portion of the circle [5].

In our example, for the intersection point *Y*, as shown in Fig. 6, the guard zonal information is ⟨*Y*, arc(*G*1, *G*2), (*I*, *J*)⟩. Hence, the guard zonal component list is updated by replacing arc(*G*1, *G*2) by ⟨arc(*G*1, *Y*), arc(*Y*, *G*2)⟩ and *IJ* by ⟨*IY*, *YJ*⟩. The next intersection point is *X* (in Fig. 7), which is between 8 and 10 (in Fig. 4(b)). 8 joins one convex point *G* and one concave point *H*. So we consider the circular arc corresponding to *G* and the line segment joining the circular arc and the concave point *H*. Again 10 joins two concave points *J* and *I*; so we consider only the line segment joining the concave points *J* and *I*. Thus, we have one circular arc and two line segments for a probable intersecting region.

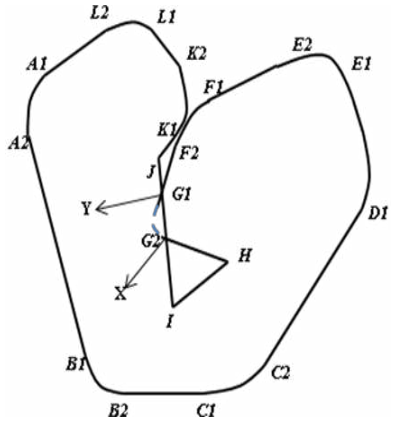

Fig. 6: Original guard zone of a simple polygon consisting of line segments and circular arcs.

After the second phase, we find *X* as ⟨*X*, (*G*2, *H*), (*I*, *J*)⟩. Now at the time of updating the guard zonal component list, as *IJ* has already been updated, we need to check whether the point *X* lies on *IY* or *YJ*. Accordingly, the guard zonal information regarding that component is updated in the list. Here for point

International Journal of Scientific & Engineering Research Volume 6, Issue 5, May-2015
ISSN 2229-5518

X we update *IY* as *IX* and *XY*, as point *X* is on *IY*.

TABLE I

THE TABLE OF EVENT POINTS, THE CONTENTS OF THE EVENT QUEUE, AND INTERSECTION LIST DURING THE EXECUTION OF THE ALGORITHM FOR THE EXAMPLE POLYGON SHOWN IN FIG. 4

Event points	Starting of line segment(s)	Ending of line segment(s)	Intersecting line segments	Event Queue	Intersection List
L	1, 12	-	-	*E*, *A*, *F*, *K*, *J*, *D*, *G*, *H*, *I*, *B*, *C*	-
E	5, 6	-	-	*A*, *Q*, *F*, *K*, *J*, *D*, *G*, *H*, *I*, *B*, *C*	*Q* (6,12)
A	2	1	-	*Q*, *F*, *K*, *J*, *D*, *G*, *H*, *I*, *B*, *C*	*Q* (6,12)
Q	-	-	6, 12	*F*, *K*, *J*, *D*, *G*, *H*, *I*, *B*, *C*	*Q* (6,12)
F	7	6	-	*K*, *J*, *D*, *G*, *H*, *I*, *B*, *C*	*Q* (6,12)
K	11	12	-	*P*, *J*, *D*, *G*, *H*, *I*, *B*, *C*	*Q* (6,12), *P* (7,11)
P	-	-	7, 11	*J*, *D*, *G*, *H*, *I*, *B*, *C*	*Q* (6,12), *P* (7,11)
J	10	11	-	*D*, *R*, *G*, *H*, *I*, *B*, *C*	*Q* (6,12), *P* (7,11), *R* (7,10)
D	6	5	-	*R*, *G*, *H*, *I*, *B*, *C*	*Q* (6,12), *P* (7,11), *R* (7,10)
R	-	-	7, 10	*G*, *H*, *I*, *B*, *C*	*Q* (6,12), *P* (7,11), *R* (7,10)
G	8	7	-	*S*, *H*, *I*, *B*, *C*	*Q* (6,12), *P* (7,11), *R* (7,10), *S* (7,8)
S	-	-	7, 8	*H*, *I*, *B*, *C*	*Q* (6,12), *P* (7,11), *R* (7,10), *S* (7,8)
H	9	8	-	*I*, *B*, *C*	*Q* (6,12), *P* (7,11), *R* (7,10), *S* (7,8)
I	-	10, 9	-	*B*, *C*	*Q* (6,12), *P* (7,11), *R* (7,10), *S* (7,8)
B	3	2	-	*C*	*Q* (6,12), *P* (7,11), *R* (7,10), *S* (7,8)
C	-	3, 4	-	-	*Q* (6,12), *P* (7,11), *R* (7,10), *S* (7,8)

At the end of Phase II, when all the guard zonal information is obtained, the algorithm reports the computed guard zone excluding the overlapped portions. Thus, the output should be in the form of list of guard zonal line segments and guard zonal circular segments after eliminating intersecting region(s). Now, here is an important observation; inside the notch area, the overlapping among the components may create two types of guard zonal loops: outer guard zone and inner guard zone. For the former case, it is a free space and not a part of the polygon. Hence, the detection of inner guard zonal loop is of utmost importance for further utilization of this free space, in terms of placement of a new subcircuit. However, both the loops are free from overlapping. For the example given, as we traverse the guard zonal components in counterclockwise manner, we can have each component in their consecutive order. According to the list, as we traverse the guard zone, we encounter the intersection points one after another, as it appears in the list. Once we encounter an intersection point, we change our path and continue the traversal along the other line segment in anticlockwise manner. Thus, this is how when we reach at the starting point of the traversal, our job is done. This process consumes O(*n*) time if the number of vertices in the polygon is *n*. As per our example, the said traversal results the outer guard zone as follows: arc(*A*1, *A*2), *A*2*B*1, arc(*B*1, *B*2), *B*2*C*1, arc(*C*1, *C*2), *C*2*D*1, arc(*D*1, *D*2), *D*2*E*1, arc(*E*1, *E*2), *E*2*F*1, arc(*F*1, *F*2), *F*2*Y*, *YJ*, *JK*1, arc(*K*1, *K*2), *K*2*L*1, arc(*L*1, *L*2), *L*2*A*1.

In this case, when we arrive at *Y* after traversing segment *F*2*Y*, we check for the line segment that intersects at *Y* other than *F*1*G*2. Here it is *JI* and the point from *Y* we find *J* in the anticlockwise direction. Thus, we move to *J* and report the segment *YJ* as the next traversed line segment in the desired guard zone excluding the overlapped regions. Sometimes there may be an overlapping at the notch region and there is a sufficient space in that notch to place a subcircuit to utilize the area more efficiently. In this case, if we follow the above procedure we compromise the possibility to find the region which is in the shape of a loop inside the notch. In that case we follow the procedure discussed below.

While traversing the guard zone, when we are at one of the intersection points, we traverse in anticlockwise direction enlisting the line segments and circular arc segments including the intersection points as well. Thus, the list starting from one intersection point and circle back to the same point needs to be eliminated from the guard zone (except the points of intersection) as it includes the inner guard zone along with

International Journal of Scientific & Engineering Research Volume 6, Issue 5, May-2015
ISSN 2229-5518

the intersection region. Then we get the resultant list for the guard zone. The inner guard zone can also be specified by sublist of the list mentioned above. If there is one such cycle starting from one intersection point and ending at the same point without having any other intersection point within it, it is an inner guard zone.

From our example polygon in Fig. 5, we have two intersection triples ⟨*Y*, (arc(*F*1, *F*2), arc(*G*1, *G*2)), (*I*, *J*)⟩ and ⟨*X*, (arc(*G*1, *G*2), *H*), (*I*, *J*)⟩. Starting from arc(*F*1, *F*2) we get the list: arc(*F*1, *F*2), *F*2*Y*, *YG*1, arc(*G*1, *G*2), *G*2X, *XH*, *HI*, *IX*, *XY*, and *YJ* as it covers all the end points of this triple and it is updated in the original guard zonal list. But here is no inner guard zone starting from *Y* and ending at *Y* as in this cycle there is another intersection point *X*. Before updating the original list we remove the sublist starting from *Y* and ending at the line segment joining two intersection points. Thus here we remove this portion: *YG*1, arc(*G*1, *G*2), *G*2*X*, *XH*, *HI*, *IX*, *XY*. Starting from arc(*G*1, *G*2) we get the list: arc(*G*1, *G*2), *G*2*X*, *XH*, *HI*, *IX*, *XY*, *YJ*. So the inner guard zone is *XH*, *HI*, *IX*, and the outer guard zone is: arc(*A*1, *A*2), *A*2*B*1, arc(*B*1, *B*2), *B*2*C*1, arc(*C*1, *C*2), *C*2*D*1, arc(*D*1, *D*2), *D*2*E*1, arc(*E*1, *E*2), *E*2*F*1, arc(*F*1, *F*2), *F*2*Y*, *YJ*, *JK*1, arc(*K*1, *K*2), *K*2*L*1, arc(*L*1, *L*2), *L*2*A*1.

Thus, the list starting from one intersection point and ending at the same point is to be eliminated from the guard zone as it includes the inner guard zone and intersection region. The inner guard zone can also be specified by sublist of the above said list. If there is one such cycle starting from one intersection point and ending at the same point without having any other intersection point in between, it is an inner guard zone.

After finding the list of intersecting line segments and arcs, the information is updated in the original guard zonal list. There is no inner guard zone starting from one intersection point (*Y*) and returning back to that very intersection point (*Y*), if in this cycle there is no other intersection point *X*, where $X \neq Y$; otherwise, there is an inner guard zone that needs to be reported separately. If there is any inner guard zone, before updating the original list the sublist starting from *Y* and ending at the line segment joining two intersection points *X* and *Y* is removed. The remaining list is reported as the outer guard zone. Thus, we obtain the outer guard zone as well as the holes in the guard zone (if any) in O(*n* log *n*) time.

TABLE II
A STUDY ON COMPARISON OF DETECTING INTERSECTION(S)

Issues considered	Naive Algorithm	Two-Phase Algorithm
Segments to be considered for checking intersections	12 (line segments) + 9 (circular arcs) = 21	Phase I: 12 (overestimated guard zonal line segments)
		Phase II: For four intersection points found in the Phase I, there are (3+3), (3+2), (3+1), and (2+1) guard zonal segments for original intersection checking.
Computation required	$(12+9)^2$ = 441	Phase I: 12 $\log_2 12$
		Phase II: (3×3) + (3×2) + (3×1) + (2×1) = 20
Time complexity	$O(n^2)$	$O(12 (\log_2 12) + cI)$ = O(*n* log *n* + *cI*), as *c* is a constant and *I* is the number of intersection points detected in Phase I.

5 EXPERIMENTAL RESULTS

In this section, we cite a relative study based on the polygon shown in Fig. 4(b), in terms of time and detecting intersections, between the naive algorithm where the intersections are identified by checking each pair of segments and our proposed two-phase method. Here we denote the number of intersections resulted in the first phase as *I* and for each probable region of intersection, we have to consider a constant number of guard zonal segments (linear or circular) for original intersection checking, each of which takes constant time using simple coordinate geometry. Note that the assumed polygon in Fig. 4(b) containing 12 edges and 12 vertices out of which nine vertices are convex and the comparison has been shown in Table II.

6 COMPLEXITY ANALYSIS

If the number of edges in the original polygon is *n*, then the number of edges by removing each of the circular arcs, i.e. to derive the extended polygon, is also O(*n*) as we require a constant number of operations for each of the convex vertices. The next step is to apply the line sweep algorithm taking the set of all line segments, original as well as derived segments after preprocessing. The algorithm starts by creating an event queue by sorting the starting and ending points of the line segments, which takes O(*n* log *n*) time using any standard sorting algorithm. Initializing the query tree takes constant time. The query tree handling consists of three operations: insertion, deletion, and interchange of node positions, each of such operations takes O(log *n*) time.

Now $m = n+I$, where *I* is the number of intersection points obtained in the first phase of the algorithm, i.e. these are the probable intersection points. Hence, the complexity of the line sweep algorithm is O(*m* log *n*) [8]. In the second phase, we deal with the segments of the original guard zone that are associated with the intersection points in the first phase. In each of the cases, we have to consider only a constant number of guard zonal segments associated with each line segment pair, which have been reported to be intersecting in the Phase I of the algorithm. Thus, total number of checking required in the second phase is *cI*, where *c* is a constant bounded within 9. The intersection points obtained in this time are the actual ones. Hence, the time complexity of the algorithm in two phases is O(*m* log *n* + *cI*), i.e. the devised algorithm is output sensitive.

7 APPLICATIONS

Now in brief we like to point out the importance and motivation of the problem as follows. Suppose, there are two (approximated) guard zones G_1 and G_2 that are computed for two 2D simple polygons P_1 and P_2, respectively, those are not shown in Fig. 7(a). Moreover, these two polygons are to be placed adjacent in realizing a larger VLSI circuit, where the

International Journal of Scientific & Engineering Research Volume 6, Issue 5, May-2015
ISSN 2229-5518

two polygons of guard zones must not overlap. Thus, there might have several 2D arrangement (or placement) of these two guard zones as shown in Figs. 7(b)-(e), out of which the placement in Fig. 7(d) takes the most reduced space (or area).

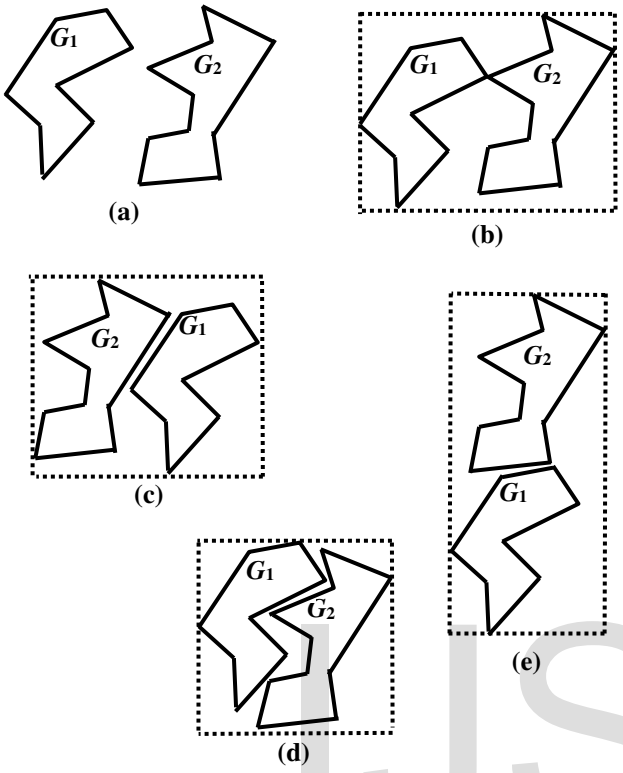

Fig.7. (a) Two (approximated) guard zones G_1 and G_2 are assumed as computed for two 2D simple polygons P_1 and P_2 (that are not shown in these figures), respectively. (b)-(e) Different 2D arrangement (or placement) of these two guard zones out of which (d) consumes the least amount of 2D space (due to better use of notches).

Though we have considered here a simple polygon, sometimes there may be more than one subcircuits whose guard zonal regions are somewhere so closed that they overlap. This compels us to compute a common guard zonal region for them removing the intersectional regions.

It may so happen that sometimes a small polygon that has been placed outside a large polygon with a sufficiently big notch in it. In this case, the small polygon could be accommodated inside the notch of the large polygonal boundary. Often this sort of placement of a small polygon inside a notch of some other polygon may provide a compact design and subsequently, space is also saved. Thus, resizing is an important problem in VLSI layout design as well as in embedded system design, while accommodating the (groups of) circuit components on a chip floor, and this problem motivates us to compute a guard zone of a simple polygon.

The guard zone problem finds another important application in the automatic monitoring of metal cutting tools. Here a metal sheet is given and the problem is to cut a polygonal region of some specified shape from the sheet. The cutter is like a ballpoint pen whose tip is a small ball of radius δ, and it is monitored by a software program. If the holes inside the notch also need to be cut, our algorithm can easily be tailored to satisfy that requirement too. The Minkowski sum is an essential tool for computing the free configuration space of translating a polygonal robot [1]. It also finds application in the polygon containment problem and in computing the buffer zone in geographical information systems [4], to name only a few.

8 Conclusion

In the case of VLSI physical design automation, given a set of isothetic non-overlapping polygonal regions and a common resizing parameter δ, the objective is to compute another set of closed regions resizing each polygon by an amount of δ. If two or more polygons are closed enough so that their guard zones overlap, indicating the violation of least separation constraint among them, then the polygons have to move relatively to overcome this breach. Resizing of electrical circuits is an important problem in VLSI layout design as well as in embedded system design, while accommodating the (groups of) circuit components on a chip floor; this motivates us to compute a guard zone of a simple polygon. In this paper, we have developed a sequential algorithm for computing the same that uses the concepts of analytical and coordinate geometry to detect overlapped region(s) within the guard zone (if any) and accordingly exclude that region to report the resulting outer guard zone. Our algorithm can also be modified to compute the regions of width r (as guard zonal distance) outside the polygon, and inside the polygon as well (if necessary), which may find several applications in practice. This work can also be extended for computing a guard zone of a three-dimensional simple solid object as a problem of probable future work.

References

[1] Bajaj C. and M.-S. Kim, Generation of Configuration Space Obstacles: The Case of a Moving Algebraic Curves, Algorithmica, vol. 4, no. 2, pp. 157-172, 1989.

[2] Chazelle B., Triangulating a Simple Polygon in Linear Time, Discrete Computational Geometry, vol. 6, pp. 485-524, 1991.

[3] Harnandez-Barrera A., Computing the Minkowski Sum of Monotone Polygons, IEICE Trans. on Information Systems, vol. E80-D, no. 2, pp. 218-222, 1996.

[4] Heywood I., S. Cornelius, and S. Carver, An Introduction to Geographical Information Systems, Addison Wesley Longman, New York, 1998.

[5] Mehera R., A. Chakraborty, P. Datta, and R. K. Pal, A 2D Guard Zone Computation Algorithm for Reassignment of Subcircuits to Minimize the Overall Chip Area, Proc. of the International Doctoral Symposium on Applied Computation and Security Systems (ACSS-2014), Apr. 18-20, 2014.

[6] Mehera R., S. Chatterjee, and R. K. Pal, A Time-Optimal Algorithm for Guard Zone Problem, Proc. of 22nd IEEE Region 10 International Conference on Intelligent Information Communication Technologies for Better Human Life (IEEE TENCON 2007), CD: Session: ThCP-P.2 (Computing) (Four pages), Taipei, Taiwan, 2007.

[7] Mehera R., S. Chatterjee, and R. K. Pal, Yet another Linear Time Algorithm for Guard Zone Problem. The Icfai Journal of Computer Sciences, vol. II, no. 3, pp. 14-23, Jul. 2008.

[8] Mehera R., A. Chakraborty, P. Datta, and R. K. Pal, A Comprehensive Approach towards Guard Zone Computation Detecting and Excluding the Overlapped Regions, Accepted for the Proc. of the International Conference

International Journal of Scientific & Engineering Research Volume 6, Issue 5, May-2015
ISSN 2229-5518

on Electrical and Computer Engineering (ICECE-2014), Dec. 20-22, 2014, Dhaka, Bangladesh.

[9] Nandy S. C., B. B. Bhattacharya, and A. Hernandez-Barrera, Safety Zone Problem, Journal of Algorithms, vol. 37, pp. 538-569, 2000.

[10] Sherwani N. A., Algorithms for VLSI Physical Design Automation. Kluwer Academic, Boston, 1993.