Data Structures for Reporting Extension Violations in a Query Range

Ananda Swarup Das[*] Prosenjit Gupta[†] Kannan Srinathan[*]

Abstract

Design Rule Checking (DRC) in VLSI design involves checking if a given VLSI layout satisfies a given set of rules, and reporting the violations if any. We propose data structures for reporting violations of the minimum extension rule in a query window of interest.

1 Introduction

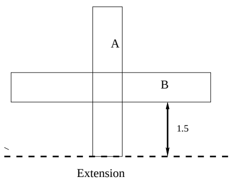

Figure 1: The minimum extension

In VLSI circuits, geometric objects (often orthogonal rectangles) of two different layers intersect to form active components. In Figure 1 the common region between the intersecting rectangles A and B is an active component. For correct working of the active component, the rectangle B must extend beyond rectangle A by some distance δ specified by design rules (1.5 units in Figure 1) and vice-versa. It is often useful for a user to find violations localized to a query window of interest. In this work we therefore intend to study some problems associated to reporting extension violations occurring inside a query window of interest.
In section 2, we discuss a simple variant of our main problem which we introduce in section 3.

2 Minimum Extension Violations in an Interval

In this paper, we assume that all the distances are Euclidean. Let $dist(a, b)$ denote the Euclidean distance between points a and b.

Problem 1 *Let P be a set of n points and I be a set of n segments, all in real line. We need to preprocess P and I into a data structure such that given a query interval $q = [a, b]$ and a parameter δ, all the triplets (p, i, e_i) where $p \in P$, $i \in I$ and e_i is an end point of the interval i such that $i \cap p \cap q \neq \phi$ and $dist(p, e_i) \leq \delta$, can be reported efficiently.*

Basic Preprocessing: Let the real line be x axis. Let $S_{\text{H}} = \{e_1, \ldots, e_{2n}\}$ be the $2n$ end points of the segments in I. Let $S_{\text{I}} = S_{\text{H}} \cup P$. Build a segment tree T_x on the interval decomposition [5] of the x axis induced by the x coordinates of the points of S_{I}. To each node $\mu \in T_x$, we assign an interval $int(\mu)$, which is the union of the elementary intervals stored at the leaves in the subtree rooted at μ. We assign a segment $i \in I$ to the node $\mu \in T_x$, iff $int(\mu) \subseteq int(i)$ but $int(parent(\mu)) \nsubseteq int(i)$. At μ we maintain a list $L_{\mu,\text{I}}$ which stores all the segments of the set I allocated to the node μ. For any point $p \in P$, let p_i be its x coordinate, we locate the leaf node $v \in T_x$ (since $p_i \in S_{\text{I}}$) which contains the value p_i. Then starting from v we allocate the point p to all the nodes which are predecessors of $v \in T_x$. For any node $\mu \in T_x$, we maintain a list $L_{\mu,\text{P}}$ which contains all the points of the set P allocated to the node μ.

Lemma 1 *The storage space required by the segment tree T_x mentioned above is $O(n \log n)$.*

Additional Preprocessing: Consider a node $\mu \in T_x$. At the node μ we build two arrays A_μ and B_μ. The array A_μ (resp. B_μ) stores the left end points (resp. right end points) of the segments present in $L_{\mu,\text{I}}$ in non-increasing order (resp. non-decreasing order). For any point $p \in L_{\mu,\text{P}}$, we find its distance from the first points of A_μ and B_μ respectively. Let the distances be d_1 and d_2 respectively. For the above mentioned point p, let p_i be its x coordinate. We then make two 2-d points (p_i, d_1) and (p_i, d_2) respectively. We then make two tuples $((p_i, d_1), address(\mu), flag)$ and $((p_i, d_2), address(\mu), flag)$ respectively. The flag is a boolean variable used to identify the array A_μ or B_μ from which the tuple originated. We follow the procedure for all the nodes $z \in T_x$. Let S_{P} be the set of all 2-d points thus formed. Build a priority search tree [6] T_{PST} using the points of the set S_{P}.

Lemma 2 *The storage space needed by T_{PST} is $O(n \log n)$.*

Query Algorithm: Given $q = [a, b]$ and a parameter δ, convert it into a three sided rectangle query of the form $q = [a, b] \times [-\infty, \delta]$. Search T_{PST} with q. For

[*]Centre for Security, Theory and Algorithmic Research, International Institute of Information Technology, Gachibowli, Hyderabad, Andhra Pradesh 500 032, India. Email: anandaswarup@research.iiit.ac.in, srinathan@iiit.ac.in.

[†]Mentor Graphics, Hyderabad, Andhra Pradesh 500 082, India. Email: prosenjit_gupta@acm.org.

each point $((p_i, d))$ reported, select the tuple for the corresponding point and visit the node $(address(\mu))$ and traverse the array A_μ or B_μ until we find the first point $e_j \in A_\mu$ or $e_j \in B_\mu$ such that $dist(e_j, p_i) > \delta$.

Theorem 1 *A set P of n points and a set I of n line segments can be preprocessed into a data structure of size $O(n \log n)$ such that given a query interval $q = [a, b]$ and a parameter δ, all the triplets (p, i, e_i) where $p \in P$ $i \in I$ and e_i is an end point of interval i can be reported in $O(\log n + k)$ time where k is the number of instances such that that $p \cap i \cap q \neq \phi$ and $dist(p, e_i) \leq \delta$.*

3 Minimum Extension Violations in a Rectangle

Problem 2 *Let H be a set of n horizontal line segments and V be a set of n vertical line segments. We need to preprocess them into a data structure such that given a query rectangle q and a parameter δ, all the triplets (h, v, p) where $h \in H$, $v \in V$ and p is an end point of either of the two segments, such that $h \cap v \cap q \neq \phi$ and $dist(h \cap v, p) \leq \delta$ can be reported efficiently.*

The X-Defect and the Y-Defect: Notice that when a horizontal-vertical intersection happens, the horizontal projection of the vertical line segment contributes to the x coordinate of the point of intersection. Similarly the vertical projection of the horizontal line segment contributes to the y coordinate of the point of intersection. We consider the intersecting pair to have X-Defect (resp. Y-Defect) if the point of intersection is at a distance less than or equal to δ from any of the end points of the horizontal (resp. vertical) segment.

Basic Preprocessing: Our outer structure is an instance T_x of the data structure of Lemma 1. T_x is built on a set of 1-dimensional points and a set of 1-dimensional intervals. The set of points is the set of x projections of the vertical segments in V and the set of intervals is the set of x projections of the horizontal segments in H. At each node $\mu \in T_x$ we maintain two lists $L_{\mu,\mathrm{H}}$, $L_{\mu,\mathrm{V}}$ which respectively stores the set of horizontal and vertical segments allocated to μ. At the node μ create an instance $T_{\mu,y}$ of the data structure of the Lemma 1. $T_{\mu,y}$ is built on a set of 1-dimensional points and a set of 1-dimensional intervals. The set of points is the set of y projections of the horizontal segments in $L_{\mu,\mathrm{H}}$ and the set of intervals is the set of y projections of the vertical segments in $L_{\mu,\mathrm{V}}$. At each node $z \in T_{\mu,y}$ we maintain two lists $L_{z,H'}$, $L_{z,V'}$ which respectively stores the set of horizontal and vertical segments allocated to z. At the node z we build two arrays $A_{z,\mu}$ (resp. $B_{z,\mu}$) which stores the y projections of the lower end points (resp. upper end points) of the vertical segments in $L_{z,V'}$ in non-increasing (resp. non-decreasing) order. Also, at any node $\mu \in T_x$, we keep an array $Y_{proj,\mu}$ which stores the y projections of the horizontal segments present in the list $L_{\mu,\mathrm{H}}$. The array $Y_{proj,\mu}$ store the values in non-decreasing order.

Lemma 3 *The storage space required by the above mentioned basic data structure is $O(n \log^2 n)$.*

Discussion: In this paper, we will discuss how to report the Y-Defects occurring inside the query rectangle. For reporting the X-Defects, similar steps are required. Consider the basic data structure of Lemma 3. Let the $q = [a, b] \times [c, d]$ and let S_{can} be the set of $O(\log n)$ nodes of T_x to which the interval $[a, b]$ is allocated. For the rest of the paper, we will consider u to be a node in S_{can}. Let h and v respectively be two horizontal and vertical line segments allocated to a node m where m is a descendant of $u \in T_x$. Let h and v intersect each other and p be the point of intersection. Then $p = (p_x, p_y) = (v_x, h_y)$ where v_x and h_y are the x and y projections of v and h respectively. It is easy to notice that p_x, the x coordinate of the point p overlaps with $[a, b]$. Therefore finding Y-Defect at any descendant (m) of $u \in T_x$ for $u \in S_{can}$ is a variant of Problem 1. Similarly finding Y-Defects at the node u itself is directly equivalent to Problem 1. However finding Y-Defects at any node m' which is a predecessor of $u \in T_x$ is not equivalent to Problem 1. This is because $int(u) \subset int(m')$. Hence there may exist a vertical segment $v' \in L_{m',\mathrm{V}}$ such that $v' \cap q = \phi$ but $v' \cap h' \neq \phi$ where $h' \in L_{m',\mathrm{H}}$ and $h' \cap q \neq \phi$.

3.1 When $m = u$ or m is a descendant of u

For any vertical segment $v \in L_{m,\mathrm{V}}$, we know its x projection v_x overlaps with $[a, b]$. Hence if there exists a horizontal segment $h \in L_{m,\mathrm{H}}$ such that $v \cap h \neq \phi$, then we are certain that the x projection of the point of intersection overlaps with $[a, b]$. However we cannot directly visit the node m because it may be quite possible that the y projection of all the horizontal segments in $L_{m,\mathrm{H}}$ is greater than d or less than c. In order to deal with the issue, we need to construct a variant of hereditary segment tree [1].

Additional Preprocessing: Consider a node $\mu \in T_x$. Remember that at node μ, we have an associated segment tree $T_{\mu,y}$. Now consider a horizontal segment $h \in L_{\mu,\mathrm{H}}$. Let S_{sec_can} be the set of all the nodes in $T_{\mu,y}$ such that $h_y \in int(z)$ for $z \in S_{sec\_can}$. Here h_y is the vertical projection of the segment h. At each node $z \in S_{sec\_can}$, we calculate the distance of h_y from the first points of array $A_{z,\mu}$, and array $B_{z,\mu}$ respectively. Let the calculated distances be d_1 and d_2 respectively. We then make two 2-d points (h_y, d_1) and (h_y, d_2) and two tuples $((h_y, d_1), address(z), flag)$ and $((h_y, d_2), address(z), flag)$ respectively. The flag is a boolean variable used to identify the array $A_{z,\mu}$ or $B_{z,\mu}$ from which the tuple originated. Let $S_{Tuple,\mu}$ be the set of all the tuples thus built for all the horizontal segments in $L_{\mu,\mathrm{H}}$. Copy the set $S_{\mathrm{Tuple},\mu}$ to all the predecessors of $\mu \in T_x$. This particular step of copying the list $S_{\mathrm{Tuple},\mu}$ to the predecessors of $\mu \in T_x$ makes our data structure a variant of hereditary segment tree. For any node $m \in T_x$, let $S_{\mathrm{desc},m} = \bigcup_s S_{\mathrm{Tuple},s}$ where s is a descendant of $m \in T_x$. Build a priority search tree $T_{\mathrm{PST},m}$ at the node m using the 2-d points of the set $S = S_{\mathrm{Tuple},m} \cup S_{\mathrm{desc},m}$.

Lemma 4 *The size of the set $S_{Tuple,\mu}$ created at the node $\mu \in T_x$ is $O(|L_{\mu,H}| \log n)$.*

Lemma 5 *The total storage space required across all the nodes in T_x because of the additional preprocessing mentioned above is $O(n \log^3 n)$.*

Query Algorithm: Given a query rectangle $q = [a, b] \times [c, d]$ and a parameter δ, allocate the segment $[a, b]$ to the nodes of T_x. Let S_{can} be the set of canonical nodes to which the segment $[a, b]$ is allocated. For any node $u \in S_{can}$, we search $T_{\mathrm{PST},u}$ with $q = [c, d] \times [-\infty, \delta]$. For any point reported, select the tuple and visit the node from which the tuple originated. Based on the flag value, visit the array A or B in that node and traverse the array until we find the first point for which there is no distance violation.

3.2 When m is a predecessor of u

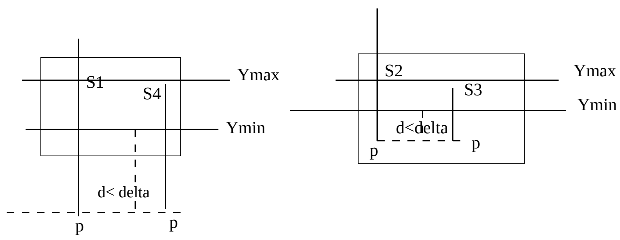

Figure 2: In this figure Y_{min} and Y_{max} are respectively the smallest and the largest y projections of all the horizontal segments intersected by the rectangle $q = [a, b] \times [c, d]$ and are allocated to the predecessors of $u \in T_x$. The figure shows how the lower end points of the vertical segments that are allocated to u and are intersecting q can cause Y-Defects inside the rectangle.

Let V' be the set of all vertical segments that are causing Y-Defects inside the query rectangle $q = [a, b] \times [c, d]$. It is easy for us to notice that $V' \subseteq \bigcup_{u \in S_{can}} L_{u,\mathrm{V}}$. Hence at any node $u \in S_{can}$ we classify, the vertical segments allocated to u into three categories. Let V_1 be the set of vertical segments which intersect q, whose both end points are outside q and at least one of whose end points is causing a Y-Defect inside q. Let V_2 be the set of vertical segments for which at least one endpoint is inside q and is causing a Y-Defect inside q. Let V_3 be the set of vertical segments for which exactly one endpoint is outside q and causing a Y-Defect inside q. In the Figure 2, S_1 is a segment of the set V_1, S_2, S_3 are segments of the set V_2 and S_4 is a segment of type V_3.

Additional Preprocessing: In the rest of the section we will discuss all the preprocessing that we need to do at every node $\mu \in T_x$ to discover the Y-Defects caused by the lower end points of the vertical segments allocated to the node μ by intersecting with horizontal segments allocated to the predecessors of $\mu \in T_x$. Similar preprocessing has to be done for finding the Y-Defects caused by the upper end points of the vertical segments allocated to the node μ.

For segments of the set V_1: Convert the vertical projections ($[v_{y_1}, v_{y_2}]$) of the vertical segments (v) in $L_{\mu,\mathrm{V}}$ into 2-d points (v_{y_1}, v_{y_2}). Build a priority search tree $T'_{\mathrm{PST},\mu}$ on the 2-d points thus constructed. The storage space required at node μ for this additional preprocessing is $O(|L_{\mu,V}|)$.

For segments of the set V_2: For any $v \in L_{\mu,V}$, build an array $info_{v,\mu}$. The size of the array should be equal to the number of predecessors of $\mu \in T_x$. For any $v \in L_{\mu,V}$ visit all the predecessors (m) of $\mu \in T_x$. Let the y projection of v be $[v_{y_1}, v_{y_2}]$. At the node m, find the smallest value (y_m) in the array $Y_{proj,m}$ such that $v_{y_1} \leq y_m \leq v_{y_2}$. Store the y_m values collected from the predecessors in the array $info_{v,\mu}$ in non-decreasing order. Calculate the distance between v_{y_1} and the first value of the array $info_{v,\mu}$. Let the distance be d. Build a 2-d point (v_{y_1}, d). Follow the same procedure for all the vertical segments in $L_{\mu,\mathrm{V}}$. Let S_μ be the set of 2-d points thus produced.

For any vertical segment in $v' \in L_{\mu,\mathrm{V}}$ such that v' does not intersect with any horizontal segment allocated to the predecessors of μ, we discard that segment. At the node μ, we then build a priority search tree $T''_{\mathrm{PST},\mu}$ on the points of the set S_μ. The storage space required for this additional preprocessing at node μ is $O(|L_{\mu,\mathrm{V}}| \log n)$.

For segments of the set V_3: Convert the vertical projections ($[v_{y_1}, v_{y_2}]$) of the vertical segments (v) in $L_{\mu,\mathrm{V}}$ into a 2-d points (v_{y_1}, v_{y_2}). Preprocess these points in an instance T'''_μ of the data structure of [2] such that given a query rectangle, all the points inside the query rectangle can be reported in $O(\log n + k)$ time where k is the output size. An instance T'''_μ of the data structure of [2] will need a storage space of $O(|L_{\mu,\mathrm{V}}| \log n)$ at the node μ.

Lemma 6 *The total storage space required because of all the three above mentioned additional preprocessing across all the nodes in T_x is $O(n \log^2 n)$.*

Query Algorithm: Given $q = [a, b] \times [c, d]$ let S_{can} be the set of $O(\log n)$ nodes in T_x to which the segment $[a, b]$ is allocated. Let u be a node in S_{can}. At the node u we do the following: We visit each predecessor (m) of $u \in T_x$ and find the smallest value (y_m) and the largest value (y'_m) in the array $Y_{proj,m}$ such that $c \leq y_m \leq d$ (resp. $c \leq y'_m \leq d$). Remember that $Y_{proj,m}$ is an array that we have created during the basic preprocessing stage at the node $m \in T_x$ and it stores the y projections of the horizontal segments present in the list $L_{m,\mathrm{H}}$ in non-decreasing order. Store all the y_m and y'_m values thus collected in two arrays $Proj_{min}$ and $Proj_{max}$ in non-decreasing and non-increasing order respectively.

Finding segments of type V_1 at the node u: Notice that any vertical segment $v \in L_{u,\mathrm{V}}$ such that $v \in V_1$ will intersect any horizontal segment $h \in L_{m,\mathrm{H}}$ such that $h \cap q \neq \phi$ and m is a predecessor of $u \in T_x$. Let Y_{min} be the first value of the array $Proj_{min}$. Notice that the lower end point p of v can cause a Y-Defect inside q iff $dist(v_{y_1}, Y_{min}) \leq \delta$ where v_{y_1} is the y projection of the point p. Hence we select the first value (Y_{min}) of the array $Proj_{min}$ and calculate the distance $dist(c, Y_{min}) = w$. If $w < \delta$, we search $T'_{PST,u}$ with the query $[Y_{min} - \delta, c) \times (d, \infty]$. Any point (v_{y_1}, v_{y_2}) thus reported is the y interval for a vertical line segment (v) such that $v \cap q \neq \phi$ and both the end points of v are outside q. Also the lower end point p of v has a Y-Defect with the horizontal segment (h) whose y projection is equal to Y_{min} and $h \in L_{m,\mathrm{H}}$ where m is a predecessor of $u \in T_x$. For each point (v_{y_1}, v_{y_2}) thus reported, we traverse the array $Proj_{min}$ until we discover a value $y_j \in Proj_{min}$ such that $dist(v_{y_1}, y_j) > \delta$. For each value $y_m \in Proj_{min}$ such that $dist(y_m, v_{y_1}) \leq \delta$ thus discovered in the above step, we visit the predecessor node m of $u \in T_x$ and traverse the array $Y_{proj,m}$ starting from y_m until we find a value $y_c \in Y_{proj,m}$ such that (a) $y_c > d$ or (b) $dist(y_c, v_{y_1}) > \delta$.

Finding segments of type V_2 at the node u: Notice that any vertical segment $v \in L_{u,\mathrm{V}}$ whose lower end point p is inside the rectangle q but the upper end point r is outside q can have an intersection with any horizontal segment inside q if and only if v intersects the horizontal segment h whose y projection is the first value (Y_{max}) of the array $Proj_{max}$. On the other hand, if both the end points of v are inside the rectangle q, we are quite certain that v intersects some horizontal segment h inside q. Remember that if there exists a vertical segment in $L_{u,\mathrm{V}}$ which does not intersect with any horizontal segment allocated to the predecessors of u, we discard that segment. Let $v \in L_{u,\mathrm{V}}$ be a vertical with its y projection equal to $[v_{y_1}, v_{y_2}]$ such that $c \leq v_{y_1} \leq Y_{max} \leq d \leq v_{y_2}$. Notice that we are sure that v causes an intersection with some horizontal segments (h) such that $h \cap q \neq \phi$ and the horizontal segments are allocated to the predecessors of $u \in T_x$. Let y_s is the smallest y projection of all the horizontal segments being intersected by v inside q. Notice that y_s will be the first value of the array $info_{v,u}$ and $Y_{min} \leq y_s \leq Y_{max}$. Remember that $\mathrm{info}_{v,u}$ is an array that we have created for the segment v at the node u during the additional preprocessing stage for the segments of the set V_2. The lower end point p of v can cause Y-Defect inside q iff $dist(v_{y_1}, y_s) = d \leq \delta$. Remember that we have the 2-d point (v_{y_1}, d) in the priority search tree $T''_{\mathrm{PST},u}$. We therefore search $T''_{\mathrm{PST},u}$ with $[c, Y_{max}] \times [-\infty, \delta]$. For any point (v_{y_1}, d) reported, v_{y_1} is the y projection of the lower end point of some vertical segment $v \in L_{u,\mathrm{V}}$. Take the array $info_{v,u}$ and the vertical projection of the vertical segment v ($[v_{y_1}, v_{y_2}]$). Traverse the array $info_{v,u}$ until we find a value y_j such that $dist(v_{y_1}, y_j) > \delta$ or $y_j > d$. For each $y_i \in info_{v,u}$ such that $dist(v_{y_1}, y_i) < \delta$ and $y_i \leq d$, visit the node $i \in T_x$ from which the value y_i originated. At node i traverse the array $Y_{proj,i}$ starting from the location of the value y_i until we find a value $y_c \in Y_{proj,i}$ such that one of the following three conditions get satisfied: (a) $dist(v_{y_1}, y_c) > \delta$ (b) $y_c > d$ (c) $y_c > v_{y_2}$.

Finding segments of type V_3 at the node u: Calculate the distance $dist(c, Y_{min}) = w$. If $w < \delta$, search T'''_u with the query $[Y_{min} - \delta, c) \times [Y_{min}, d]$. For any point reported proceed as discussed for the segments of type V_1.

Theorem 2 *A set H of n horizontal line segments and a set V of n vertical line segments can be preprocessed into a data structure of size $O(n \log^3 n)$ such that given a query rectangle $q = [a, b] \times [c, d]$ and a parameter δ, all the triplets (h, v, p) where $h \in H$ $v \in V$ and p is an end point of either of the segments can be reported in $O(\log^2 n + k)$ time where k is the number of instances such that that $h \cap v \cap q \neq \phi$ and $dist(h \cap v, p) \leq \delta$.*

References

[1] B. Chazelle, H. Edelsbrunner, L.J. Guibas and M. Sharir. *Algorithms for Bichromatic Line Segment Problems and Polyhedral Terrains*, Algorithmica 11: 116-132 Springer Verlag (1994)

[2] P.K. Agarwal, and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, Contemporary Mathematics, 23, 1999, 1–56, American Mathematical Society Press.

[3] P. Gupta. Algorithms for range-aggregate query problems involving geomteric aggregation operations. *Proceedings, International Symposium on Algorithms and Computation*, Springer Verlag LNCS, Vol. 3827, 2005, 892–901.

[4] T.G. Szymanski, and C.J. van Wyk. Layout analysis and verification, in *Physical Design Automation of VLSI Systems*, B. Preas and M. Lorenzetti eds., Benjamin/Cummins, 1988, 347–407.

[5] M. de. Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications, Springer, Verlag, 2000.*

[6] E.M. McCreight. Priority Search Trees, *SIAM Journal of Computing, 14 (2), 1985, 257-276.*