A Depth-First-Search Controlled Gridless Incremental Routing Algorithm for VLSI Circuits

Hasan Arslan and Shantanu Dutt
Dept. of ECE, University of Illinois-Chicago
{harslan, dutt}@ece.uic.edu

Abstract: In the engineering change order (ECO) process, engineers make changes to VLSI circuits after their layouts are completed in order to correct electrical problems or design errors. As far as routing is concerned, in order to capitalize on the enormous resources and time already spent on routing the circuit, and to meet time-to-market requirements, it is desirable to re-route only the ECO-affected portion of the circuit, while minimizing any routing changes in the larger unaffected part of the circuit in order to preserve its electrical properties. In this paper, we develop a novel algorithm to find incremental routing solutions using a gridless framework for VLSI circuits that require variable width and variable spacing on interconnects. The basic idea in our algorithm is to route the new or ECO-modified nets by minimally re-arranging, if necessary, some portions of some existing nets using a novel DFS controlled process that does not allow the perturbed existing nets' lengths and topologies to change beyond pre-set limits. With these constraints, it explores a number of low-cost ways of re-routing the portions of these nets within the available routing resources (2 metal layers only). Experimental results show that within the above constraints our incremental router succeeds in routing more than 98% of ECO-generated nets, and also that its failure rate is 5 to 12 and 2.4 to 9 times less than that of previous incremental routing techniques Standard (Std) and Rip-up&Reroute (R&R), respectively. It is also able to route most of the wide nets using a reasonable number of vias and with near-minimal net lengths.

1 Introduction

An engineering change order (ECO) is a request to make design changes which are made typically late in the design process in order to correct logical and/or technological problems in the circuit. Engineers usually want to modify the existing solution incrementally and keep the design as close as possible to the existing one. It requires very efficient incremental design algorithms and methodologies to make such changes. Incremental routing is an integral part of any incremental physical design flow. Redoing the routing for all the nets to tackle the incremental changes is too time-consuming and can play havoc with time-to-market needs and may introduce new errors into the design. Moreover, an entirely different layout may completely invalidate the detailed timing results, which is undesirable. It is thus desirable to re-route only the ECO-affected portion of the circuit, while minimizing any routing changes in the much larger unaffected part of the circuit. ECO incremental routing is challenging in two aspects. First, there are a large number of existing interconnect which become obstacles in the region. Second, advances in circuit designs require variable width and variable spacing on interconnects [1]. Thus, not only a gridless incremental re-routing algorithm is needed but also such an algorithm needs to be fast and to effectively use available routing resources in order to avoid time- and area-expensive processes like re-floorplanning, re-placement, channel expansion and increasing the number of routing/metal layers to complete the required routings.

Past work tackling this important problem include [6, 7, 12, 13]. In [6] only the portions of nets whose pin connections change are re-routed. This re-routing is done in a standard single-net routing mode in which global routing is followed by detailed routing in the available routing space without disturbing existing net routes; we term this incremental routing approach as `Standard` (*Std*) in the rest of the paper. Limited search space is the major disadvantage of this scheme. The rip-up and reroute (*R&R*) approach to incremental physical design is presented in [7]. First they route the new nets without removing any existing nets. When some nets cannot be routed, some of the existing nets are ripped-up to free up routing resources. The routing is then re-done for the new nets followed by the ripped-up nets. If *R&R* fails to route all the nets, the given floorplan and placement configurations are needed to be modified. The main disadvantage of a *R&R* scheme is that the routing is no longer truly incremental, as existing nets are ripped out and there is little control on how they are re-routed. Also most algorithms proposed for rip-up and reroute [3, 4, 5] assume that there exists a underlying uniform routing grid and all net segments can be simplified as a zero width lines centered on the grid. However, this assumption does not hold anymore in variable width and variable spacing routing. An incremental routing algorithm for FPGAs that uses a novel *bump-and-refit* (B&R) approach which routes the new nets by "bumping" less critical existing nets without changing their topologies was proposed in [13] and significantly extended for ECO routing and for FPGAs with complex switchboxes in [12]. Unlike [6], this approach eliminates the net ordering problem by changing the track assignments for previously routed nets and gives optimal solution in the number of used tracks.

In [2] efficient techniques are presented for obtaining a non-uniform routing grid from a given VLSI routing in order to perform incremental routing for ECO. This is an important issue for incremental routing in VLSI circuits since it is impractical to store uniform dense routing grids for current very deep submicron technology. In this paper we present a somewhat different non-uniform routing grid that can be extracted locally and quickly.

In this paper we present an incremental routing algorithm for complex VLSI circuits which have variable width and variable space requirements. We develop a well-controlled depth-first-search (DFS) directed rip-up and rerouting approach which routes the new nets by minimally perturbing existing nets. Our approach does not rip-up and reroute the bumped nets in their entirety, but only reroute their bumped segments (or their subparts) on other vacant or minimally occupied locations within a tight bounding box. If such a rerouting of the bumped segment fails, its bumping is retracted in the high-level DFS control and another position of the bumped net segment is explored. Such a DFS control keeps the topologies and lengths of perturbed nets close to their original topologies and lengths.

The goals of our work are to develop incremental routing algorithms that: (1) are orders of magnitude faster than complete re-routing; (2) complete the required incremental routing in the available routing resources if such a solution exists (this will minimize the need for area- and time-expensive fall-back strategies); (3) complete the routing without significantly changing electrical properties (e.g., power, delay) of existing nets. Experimental results presented in Sec. 6 show that our incremental router has significant advantages over existing incremental routing methods with respect to the above metrics.

The rest of the paper is organized as follows. In Sec. 2, we define the incremental routing problem for VLSI circuits. Definitions pertaining to our incremental routing algorithm are given in Sec. 3. Gridless VLSI routing and a new technique for non-uniform grid extraction for a given routing are discussed in Sec. 4. Section 5 explains all facets of our incremental routing algorithm. Experimental results comparing our incremental router to the type of incremental routers proposed in [6, 7] that we have implemented, are given in Sec. 6. We conclude in Sec. 7.

2 Incremental Routing for VLSI Circuits

The incremental routing problem that we will solve in this paper is for gridless VLSI circuits having variable width and variable space requirements. While getting a solution for the ECO-generated net, it might overlap with some existing nets or the horizontal/vertical spacing between two wire segments may violate the wire separation requirement (a situation we call *bumping*). Hence to be able to route it, some properties (track position, number of vias, net length, etc.) of some existing nets needs to be changed. Due to shifting existing nets to other locations, some segments of shifted nets become shorter while others become longer. Also, the changes on one layer can cause changes on other layers. To avoid these problems we should aim to route new nets such that there is minimal modification on the existing layout. *The key problem in incremental routing is to preserve as much of the previous routing results as possible, while accommodating the new routing requests without wire spacing violations*.

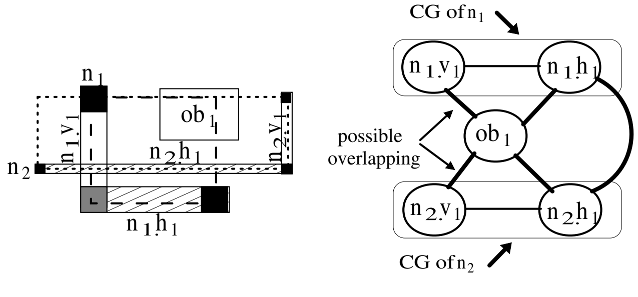


Figure 1: *(a) Net Routing and their BBs. (b) Contiguity-Graph (CG) of n_1 and n_2 and the overlap graph (OG) representing possible overlappings if any segment is moved to other locations.*

The overall quality of incremental routing solution can be measured in terms of total wire length, number of vias, number of layers and among of changes made to topologies and net lengths of existing nets. These metrics can be minimized by modeling them as either optimization metrics (e.g., wire length, number of layers) or constraints (e.g., a maximum number of vias per net, routing new as well as perturbed/bumped existing nets within certain bounding boxes). Our incremental router will optimize the number of routing layers by exploring a richer solution space (within the original number of routing layers) than previous methods. This exploration is achieved by a controlled DFS wherein we backtrack and search for alternative solutions when the current search violates given constraints or is not able to find a feasible solution. Our algorithm will minimize the other metrics by modeling them as and satisfying the following constraints.

a) Via Constraint: More than 70 % of VLSI circuit nets are two terminal nets [14]. Previous studies showed that most two pin nets can be routed by using a maximum at four vias [8, 14]. Our algorithm will route nets with this constraint. Bounding the number of vias per net permits efficient routing of all the nets and allows for more precise delay estimation at higher levels of the design.

b) Bounding Box Constraint: Each ECO-generated net as well as each existing net that is perturbed will be routed by our algorithm within a somewhat larger region (the *routing bounding box [R-BB]*) than the minimal bounding box (BB) that contains the two pins of the net. First, we try to route each net in its BB. If such a solution cannot be found, we allow some flexibility in order to have a somewhat larger solution space and thereby reach an overall better solution across all nets. In order to connect pin P located at $(x1,y1)$ and pin Q located at $(x2,y2)$ where $(x2 > x1)$ and $(y2 > y1)$, the R-BB for the 2-pin net will be $(x1-\Delta, y1-\Delta)(x2+\Delta, y2+\Delta)$ where Δ is a small relaxation.

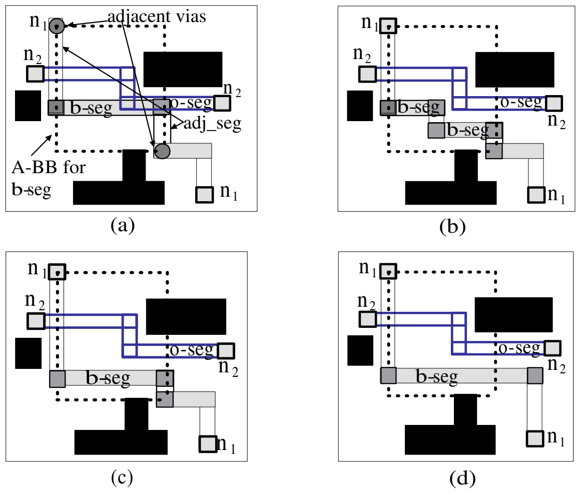


Figure 2: *Routing new net n_2 creates an overlapping problem. (a) The h-segment of net n_1 (b-seg) is overlapped by an h-segment of new net n_2 (o-seg). Dashed lines shows the* A-BB *for the b-seg; adj-vias are shown as circles. (b) A solution which is invalid because of via constraints. (c) A valid solution for the b-seg of n_1. (d) A refinement algorithm improves the solution of part [c] by reducing the number of vias.*

3 Incremental Routing Concepts

Before going into the details of our incremental routing algorithm, we define a few terms. Each net is routed using up to five connected segments alternating between vertical and horizontal directions. We call a horizontal segment an *h-segment* and a vertical segment a *v-segment*. We will denote the k^{th} *h-segment* of net n_i as $n_i.h_k$ and the j^{th} *v-segment* of net n_i as $n_i.v_j$. The connected graph consisting of horizontal and vertical segments of a net is defined as a *contiguity graph (CG)* with each net segment represented as a node in this graph. The segments which are adjacent in the layout (connected through via) have an edge between them in CG. The CG for a circuit in Fig. 1(a) is shown in Fig. 1(b).

The *overlap graph OG(V,E)* is a graph representation of the circuit routing where the set of the nodes $V = OB \cup \{S_1, S_2, S_3, \ldots, S_i, \ldots, S_k\}$, S_i is a routed segment (h-segment/v-segment) (see Fig. 1) and OB is the set of obstacles such as cells of some nets or pins of a cell. If the two segments $n_k.S_i$ and $n_l.S_j$ of the nets n_k and n_l whose *BB*'s intersect are parallel to each other and there exists a line perpendicular to the them which cuts both of them, then there is an edge between them (this indicates a possible overlapping between $n_k.S_i$ and $n_l.S_j$ if either one of them is shifted along the perpendicular line). If any $ob_i \in OB$ is in the *BB* of net n_k and a line perpendicular to segment $n_k.S_j$ cuts ob_i, then there is an edge between them.

When routing a net n_i, if a segment $n_i.S_k$ overlaps existing segments of other nets, $n_i.S_k$ is denoted as an *o-seg* and the existing segments overlapped are termed as *b-seg*s. The segment(s) of same nets adjacent to a *b-seg* are termed "adjacent segment(s)" (*adj-seg*) and the via/pin which is connected to an *adj-seg* but not the corresponding *b-seg* is denoted as an "adjacent via" (*adj-via*); see Fig. 2. While finding a solution for shifting a *b-seg*, we move it to different locations in the rectangular region *A-BB*, which is the smallest rectangular box containing all the adj-vias of the *b-seg*; see Fig. 2.

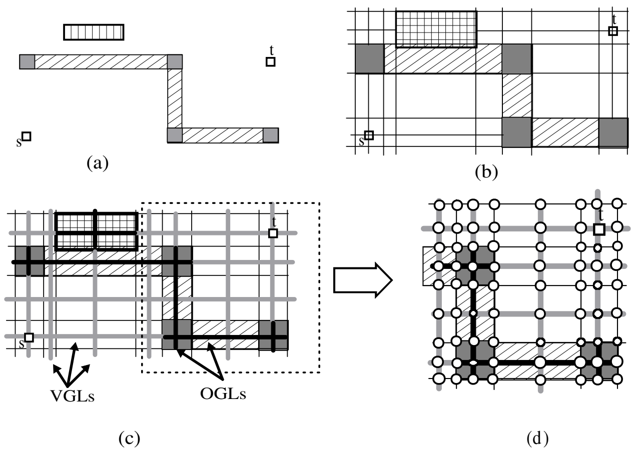


Figure 3: *Non-uniform grid graph generation by using* type 1 *and* type 2 *(OGL and VGL) grid lines.* Type 2 *grid lines are shown as thick lines. (a) The boundary of objects without expansion. (b)* Type 1 *grid lines generated from the boundary of expanded rectangles shown shaded and from x,y locations of the new net's s and t pins. (c)* Type 2 *grid lines are added; VGLs are lightly shaded thick lines, OGLs are dark thick lines and* type 1 *lines are shown as thin lines. (d) Example of the non-uniform graph NUG constructed for the region highlighted in (c).*

4 Gridless Routing and Non-uniform Grids

In gridless VLSI routing, the layout is conceptually populated with numerous feasible points where the center-line of a net path can pass through. The feasible points and their neighborhood information can be abstracted as a grid graph, called *the non-uniform grid graph (NUG)*, so that the optimal path can be found using a shortest path algorithm. Initially a grid graph is built based on the obstacles and existing nets in the routing region, and then a search algorithm (described later) is applied on the graph to find a route for the new net. When a new net is to be routed, a local grid graph is created in the R-BB of the source pin s and target pin t. The first problem that we need to address is how to construct a grid graph that is much sparser than the manufacturing grids and yet contains the optimal route if one exists. Many algorithms simplify the grid graph at the expense of very costly pre-construction and representation [9, 10, 11]. Therefore, their usefulness is quite limited for large designs as in the case of most ECO routing scenarios.

When routing a new net n_i, the minimum space between the new net's segments and the rectangular obstacles in the *BB* of the n_i, that are adjacent to n_i in the OG, must satisfy design spacing rules. In order to satisfy the layout design rules we create an obstruction zone around each obstacle net segment where the centerlines of wires cannot placed [1] [2].

Our *NUG* is an orthogonal grid graph, consisting of x and y grid lines. There are two types of grid lines depending upon how they are determined. *Type 1* grid lines are obtained from the vertical and horizontal boundary lines of expanded routed net segments and obstacles in the *BB* of the new net. In addition to that, it includes horizontal and vertical lines passing through the center of s and t pin points of the new net (see Figs. 3a-b). We define *type 2* grid lines as the lines between the *type 1* grid lines. We define *occupied grid lines (OGL)* as those *type 2* grid lines part of which are occupied by already routed net segments. The width of these grid lines are determined by using the vias size of the nets on them. We also define *variable space grid lines (VGL)* as vacant grid lines between two *OGL* lines if there is enough space between them (as determined by via- and wire-spacing, and wire-width design rules) to place a wire; see Fig. 3 and Fig. 5. The intersection of grid lines give us the grid points or nodes as shown in Fig. 3d. The pseudo code of *NUG* is given in Fig. 4. If d_i is the number of nets intersecting the BB of n_i, there will be $5d_i$ (a net can be routed using a maximum of 5 segments) comparisons to decide actually which segments of intersecting nets will be expanded. The time to scan all these segments and extract their boundaries is $\Theta(d_i)$, and the time for sorting these boundaries to get type 1 grid lines is $\Theta(d_i \log d_i)$. Generating the type 2 grid lines between each type 1 grid lines takes $\Theta(d_i)$ time.

The non-uniformly gridded nature of *NUG* makes it very easy to come up with a routing space representation that is both highly compressed in storage and efficient in searching for solution paths. This is in contrast to a uniform grid graph that has a very dense grid of all routing grid-lines.

5 A DFS-Based Incremental Routing Algorithm

When a new net n_i is routed, we create a non-uniform grid graph *NUG* for n_i to find a path from source pin s to target pin t of the new net. Also an *NUG* is created to find a path for bumped segments *b-seg* and its *adj-segs* of an overlapped net.

When bumping occurs, the effected segments (*b-seg* and its *adj-segs* —in the rest of the paper *b-seg* will be used to refer to itself and its *adj-segs*) are ripped up and after the *NUG* is extracted, a path is searched between the *adj-vias* of the *b-seg* to reroute them in its *A-BB*; see Fig. 5a. The ripup and rerouting process which is performed to get a solution path for the *b-seg* is equivalent to moving the *b-seg* or its subparts to different grid lines; see Fig. 5c. To obtain a valid solution between *adj-vias*, when the *NUG* is created,

[1]If wire width of net n_i is w, each obstacle ob_j in the BB of net n_i is expanded from w_j to $(w_j + w/2 + w_{s_{ij}})$ where $w_{s_{ij}}$ is the space requirement between net n_i and obstacle ob_j, and w_j is the width of obstacle ob_j.

```
Algorithm Extract-Non-Uniform-Grid($n_i$, OG)
  Input: $n_i$ is either a new net or a bumped portion of an existing net;
      OG is overlap graph;
  Output: Non-uniform grid;
Begin
    Get A-BB of bumped segment (b-seg) of net $n_i$;
    for each net (obstacle) $n_k(ob_k)$ adjacent to $n_i$ in the OG do begin
      if(segment $S_j \in n_k$ and $S_j \cap A\text{-}BB \neq \emptyset$
        or obstacle $ob_k \cap A\text{-}BB \neq \emptyset$) then
        expand $S_j$ or $ob_k$ according to net $n_i$'s width and spacing rule;
      endif
    endfor
    Generate type 1 grid lines from the boundary of expanded $S_j$ or $ob_k$
    Generate type2 grid lines between each pair
        of adjacent type 1 grid lines if spacing rules permit;
    Construct non-uniform grid $NUG$;
    return($NUG$);
End. /* Extract-Non-Uniform-Grid() */
```

Figure 4: *Pseudo code for extracting the non-uniform grid graph.*

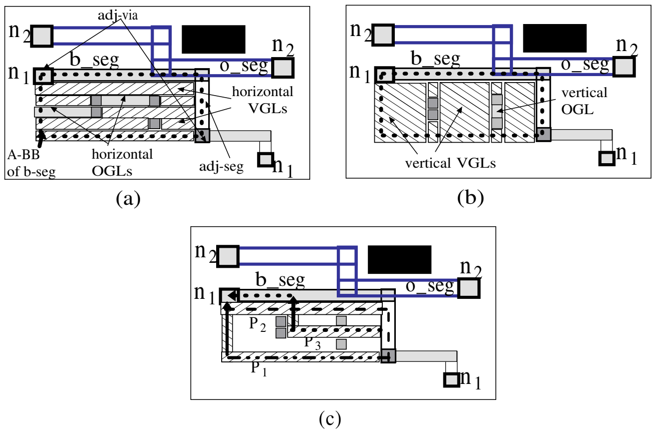
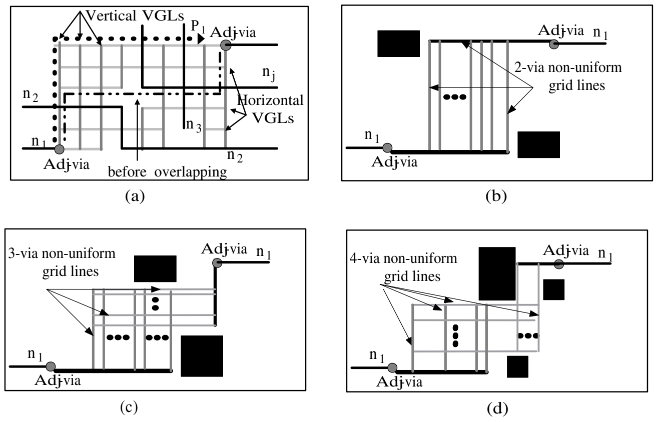


Figure 5: VGLs *and* OGLs *inside the* A-BB *of a bumped segment b-seg, obstacle is shown with dark color. (a) Possible* VGLs *in paths for the* b-seg *(and its adj-segs) within the* A-BB *of the* b-segs. *(b) Possible vertical* VGLs *to get path(s) for* b-seg. *(c) Different combinations of* VGLs *which produce solutions with different number of vias for finding path for the* b-seg.

we first try to find a path consisting of *VGLs* which are consecutively connected and require fewer vias. As seen in Figs. 5a-b, there can be different *VGLs* in the *A-BB* of the *b-seg* and might be used for finding a solution for its shifting. As shown in Fig. 5c, if we select the *VGLs* on path P_2, we get a solution requiring two vias. If *VGLs* on path P_3 are selected, then we get a 3-via solution, and selecting the *VGLs* on path P_1 requires only one via. Since these new paths will be in the *A-BB* of the *b-seg*, the net length will not be changed.

Our goal is to find a path consisting of *VGL* and *OGL* segments (preferably *VGL* alone to avoid modifying previous nets) by rerouting bumped segments between their *adj-vias* while keeping the number of vias close to optimal for each net. In order to find such paths, we use the approach described in [8]. The approach of [8] was developed for grid-based routing, and we extended it for gridless routing using *NUGs* with variable width and variable space requirements.

There are two phases in our algorithm to find a valid path for *b-seg*. Like the *Std* approach, in the first phase we try to find a path without bumping of any existing nets. If this is not possible, we allow modification of existing nets to find a solution in the



Vertical VGLs
P1
Adj-via
nj
n2
Horizontal VGLs
n3
n1
before overlapping
2-via non-uniform grid lines
3-via non-uniform grid lines
4-via non-uniform grid lines
(a)
(b)
(c)
(d)

Figure 6: *Different path finding scenarios with different numbers of vias. (a) One-via solution for the* b-seg *of net* n_1 *without overlapping with existing segments, after new net* n_j *is routed. Gray lines show the non-uniform gridlines inside the* A-BB *of b-seg of net* n_1. *(b-d) For simplicity only the possible routing paths are shown for two (b), three (c) and four (d) via routing solutions.*

second phase. If both phases fail, we increase the R-BB (subject to maximum of 25% beyond the pin BB) of the new net and redo the two phases until the R-BB cannot be expanded either because it is blocked by pins or the layout boundary or the maximum limit has been reached.

5.1 Phase 1: Bumpless Routing

In this phase, we only try to find an appropriate path $P_k \in PV$ for the *b-seg* where *PV* is the set of paths between *adj-vias* that do not bump any existing net. In this case, the number of vias used in the path P_k becomes the dominant cost. Recall that we can use a maximum of four vias for the entire net. We thus generate paths in order of increasing via numbers.

Figure 6b-c shows how solution paths are explored for different number of vias. If not prevented by obstacles (vias, prerouted nets, cells), then one via routes are constructed along the bounding box perimeter. At most two such routes exist. If one via routes have not been found, then two via routes are constructed by selecting three intersection line segments which connect the two *adj-vias* of the *b-seg* along grid lines that are not occupied by obstacles or segments of other routed nets. Illustration of example two via routes is shown in Fig. 6b. If there is no 2-via solution, four and five intersection line segments along grid lines are constructed to route the nets by using 3 and 4 vias, respectively, as shown in Figs. 6c-d.

5.2 Phase 2: DFS-Controlled Segment R&R

If there does not exist a path $P_k \in PV$ for the *b-seg*, we explore the paths $P_k \in PO$ where *PO* is the set of the paths that overlap bumping segments of existing nets. These bumped segments (*b-seg*) of existing nets in turn will have to be shifted in their own *A-BB* between their *adj-vias*, giving rise to a sequence of shifting and path searching for each *b-seg*. Note that we do not ripup and reroute the entire bumped net as in the case of [7]. Here, we are only ripping-up and re-routing the *b-seg* of the bumped net; the rest of the bumped net is treated as an obstacle until we find a solution for the *b-seg*. Further, the *R&R* of the *b-seg* is performed using a DFS-controlled process that retracts possible *R&Rs* of *b-segs* if finding new routes for them causes a lot of subsequent *R&Rs* of other *b-segs* and/or these routes are blocked due to surrounding obstacles. This prevents the *R&R* process from going out of control as is possible in traditional *R&R* [7]. It also preserves the net

```
Algorithm DSR-Routing($n_i$, $OG$)
Begin
    $NUG$=Extract-Non-Uniform-Grid($n_i$,OG);
    result=Route-Without-Overlapping($n_i$,$NUG$); /* Phase-1 */
    if (result==success) then return(success);
    $C_p$=Get-Candidate-Paths($n_i$);
    while($C_p \neq \emptyset$) do /* until solution found
                                    or all candidates are tested */
        $B_p$=Get-Next-Best-Path($C_p$);
        $K$=set of nets will be bumped on path $B_p$;
        for each $n_j \in K$ do begin
            remove b-seg of $n_j$.
        Endfor
        succ-routed-child=0;
        for each $n_j \in K$ do begin
            mark $n_j$ as visited;
            result=DSR-Routing($n_j$, $OG$)
            mark $n_j$ unvisited;
            if (result==fail) then break;
            else
                succ-routed-child ++;
        Endfor.
        if (succ-routed-child== $|K|$) then return(success);
    endwhile;
    return(fail); /* no solution */
End. /* DSR-Routing () */
```

Figure 7: *The DSR incremental routing algorithm.*

lengths and minimizes vias in the shifted routes thus minimally perturbing existing nets and thus preserving electrical properties to a large extent. This phase is the crux of our incremental routing algorithm. We thus use the acronym *DSR* for our algorithm, which stands for *Depth-first search controlled Segment R&R*.

Finding a path for a particular *b-seg* can initiate a set of other *b-seg* shiftings and path searchings which finally terminate when a $P_k \in PV$ is found for each *b-seg*. Essentially, we are performing a depth first search (DFS) of sequential shiftings, to find a solution for the original *b-seg*, i.e., *a DFS path terminates in failure if the route selected for the current* b-seg *overlaps already bumped nets in the current path or overlaps obstacles*. If a particular path $P_k \in PO$ fails in this manner, the search backtracks and tries another unexplored path P_l for the *b-seg*. The final set of successful shiftings resulting from the initial overlaps of the path chosen for new net n_i takes on a directed-acyclic graph (DAG) structure termed a *Path DAG (P-DAG)*, where each interior node is a *b-seg*, each edge is a path $P_k \in PO$ and leaf nodes are the *VGLs* which do not cause any overlap with existing nets; see Fig. 8c.

As seen in Fig. 8b, the path found as a possible solution for the *b-seg* of net n_1, overlapped by the *o-seg* of new net n_j in Fig. 8a, overlaps with the horizontal segment $n_2.h_2$ of net n_2. In order to resolve this overlap, as a next step, the *b-seg* of n_2 is shifted in its *A-BB* and this shifting creates another overlap with segment $n_3.v_1$ of net n_3. Due to bumping of obstacles and ancestor nets among candidate paths for $n_3.v_1$, *DSR* cannot find a solution for it and its adjacent segments. In this case, it backtracks to try another path for $n_2.h_2$ with a different numbers of vias (see Figs. 8b-c). However, testing other paths for segment $n_2.h_2$ also fails and *DSR* backtracks to the *b-seg* of n_1 to try another path for it. As seen in Fig. 8d, when we try to route the *b-seg* of n_1 by using another path which requires two vias, another segment $n_2.h_1$ is overlapped. However, in this case, we get a solution for this segment without overlapping any other existing net segment by routing it in the second try on a 2-via path on available *VGLs*.

In phase 2, there can be many candidate paths for a *b-seg*.

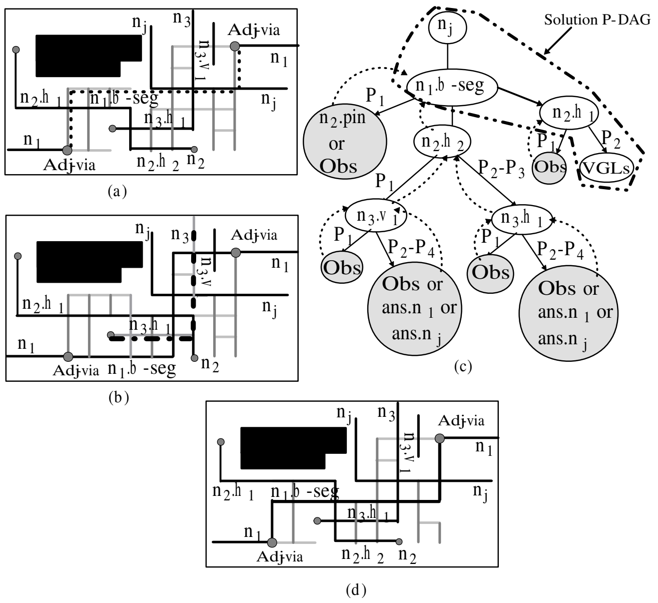


Figure 8: *Routing example for Phase-2 (finding path by allowing modification on existing nets). (a) Dashed lines shows the ripped up portion of overlapped segments of net n_1 by new net n_j. (b) Here shifting the b-seg of net n_1 causes other overlaps. (c) The DFS rooted at net n_j. Shaded circles show fail shiftings which occur when the chosen route either bumps obstacles or ancestor nets. Dashed lines show backtracking after all paths tried for the particular b-seg have failed. P_i denotes a path which uses i vias. The rightmost transitions in the DFS tree show a successful solution represented by a sample P-DAG. (d) Final routing result.*

These *m* candidate paths for the *b-seg* can be found (within its *A-BB*) in random, sequential (the first *m* paths in increasing distance, say, from the left *adj-via*) and equally spaced from each other, in each of the 4 categories, 1 to 4-via paths. The cost for these *m* paths are determined using a cost function described shortly and of these the *x* least cost paths are tried sequentially in increasing cost order to determine if there is a solution *P-DAG* rooted at the current *b-seg*. If none of the *x* paths are successful, the algorithm retracts the *R&R* of the current *b-seg* and tries another path for its parent *b-seg* node. After significant testing, the value of $x = 10$ and the randomized initial path set selection approach gives us the best solutions in terms of both quality and runtime.

It will be time-efficient if some suitable "cost" measure can be used to determine which candidate paths are more likely to be successful, so that fewer P-DAGs are searched. We use the following cost function for a path P_k:

$Cost(P_k)=[\sum_{n_j \in Q} \alpha.D(n_j) + \beta.Via\text{-}Num(n_j)]/\sqrt{|Q|}$

where Q is the set of the nets overlapped by P_k and $D(n_j)$ is the degree of a net n_j in the *OG*. The lesser the degree of net n_j, the lesser number of segments it might overlap with other existing nets when finding a solution for its overlapped segment. The second term in the cost function represents the number of vias which were used to route the overlapped net n_j. Lesser the number of vias used during routing of net n_j, whose portions are being overlapped, greater will be the probability/flexibility of solutions for the overlapped portion of net n_j, for satisfying the via limit constraint for the entire net n_j. α and β are the weighting factors for the net degree and via number metrics, respectively. We simulated the cost function for different values of α and β. Based on the simulation results, we found that $\alpha = \beta = 0.5$ gives the best results in

```
Algorithm Standard-Routing(S)
    Input: The new netlist S will be added ;
    Output: Incremental routing result;
Begin
        Create $OG$;
        for each $n_i \in S$ do begin
            $R\text{-}BB(n_i)=BB(n_i)$; /* initialize bounding box
            repeat
                $NUG$=Extract-Non-Uniform-Grid($n_i$,OG);
                result=Route-Without-Overlapping($n_i$,$NUG$);
                if (result==success) then break;
                    else Expand R-BB($n_i$);
            Until ($R\text{-}BB$ of $n_i$ cannot be extended );
        Endfor
End. /* Standard-Routing () */
```

Figure 9: *The previously proposed standard (Std) incremental routing algorithm.*

terms of solution quality metrics as well as runtime.

If phases 1 and 2 cannot find a solution for the new net in its *BB*, we allow routes to extend outside the bounding box by increasing the R-BB until it cannot be increased further because of obstacles or reaching the layout boundary or the prescribed expansion limit. The same *R-BB* expansion is also used for the other two methods *Std* and *R&R*, with which we compare our *DSR* algorithm. An overview of our algorithm as well as *Std* and *R&R* are given in Figs. 7, 9 and 10 respectively.

6 Experimental Results

The *DSR* incremental algorithm was tested on a number of benchmarks which we created by using the Mcc1 circuit (an MCM circuit) cell distribution structure as an example; their characteristics are shown in Table 1. The number of nets in our benchmarks ranges from 97 to 951 with randomly assigned variable widths and variable spaces for each net. We also tested our algorithm on some well known standard cell benchmarks such as Primary1, Struct and S9234. However the nets in these benchmarks have uniform width and space requirements. Further, well known VLSI routing algorithms such as MARS [15] use 3 or 4 layers to route the nets for each of these circuits; only the first two layers are well used while layers 3 and 4 are sparsely populated. Hence even a simple incremental routing algorithm such as *Std* is able to route all new nets on layers 3 and 4 without modifying the existing routing. We thus generated new benchmarks with only 2 routing layers that can test incremental routers more rigorously. Furthermore, these benchmarks have nets with non-uniform width and space requirements.

To simulate ECO, we randomly removed 10% and 20% of the original nets, and added the same percentage of new nets with the same number of pins but with random pin positions, net width and space requirements. For each circuit we perform these random deletions and new net creations 20 times thus generating 20 different problem instances for both 10% and 20% net deletion and creation cases. On the average, the half-perimeter (HP) BBs of the new nets were 7% and 6% more than those of deleted nets for 10% and 20% cases, respectively. We compared our *DSR* algorithm to the two prior techniques *Standard (Std)* [6] and *Rip-up&Reroute (R&R)* [7] implemented by us. In *R&R*, we allow an existing net to be ripped up and rerouted up to r times in the process of finding a solution for a new net; we collected the results for $r = 5$, 10, 15 and 20 and these will be discussed shortly.

We ran all three methods on 2.6 Ghz Pentium Linux machines with 1GB of RAM. The results are tabulated in Tables 2 to 5. In these tables, *HP of F. net* is the average half-perimeter (HP) of the

	10%	**20%**		**10%**	**20%**		**10%**	**20%**
Circuit	**New N.**	**New N.**	**Circuit**	**New N.**	**New N.**	**Circuit**	**New N.**	**New N.**
net-97	9	19	**net-102**	10	20	**net-103**	10	20
net-115	11	23	**net-247**	24	49	**net-282**	28	56
net-391	39	78	**net-413**	41	82	**net-557**	55	111
net-700	70	140	**net-797**	79	159	**net-829**	82	165
net-961	96	190	**net-968**	95	193			
Average number of new nets							**46.36**	**93.21**

Table 1: *Characteristics of Benchmarks.*

Metrics ↓	**With 10% new nets**			**With 20% new nets**		
	Std	**R&R**	**DSR**	**Std**	**R&R**	**DSR**
Avr. Unrouted nets (%)	17.3%	7.4%	1.6%	14.4%	4.1%	1.7%
Avr. Time (sec.)	3.09	12.08	34.37	4.19	15.61	45.51
Avr. # of succ. runs out of 20	0.15	3.9	10	0.0	3.86	7.43
Avr. % of unsucc. nets per fail run	17.4%	9.2%	3.2%	14.4%	5.1%	2.8%

Table 2: *Simulation results for 10% and 20% new nets. R&R results are for r =5.*

BB of unrouted nets and *F.net.W.* is the average width of unrouted nets. *DSR* was able to route almost all the nets without sacrificing quality metrics such as the average number of used vias and the total net length; note that we are restricting all routes to be contained within two routing layers to test the incremental routing methods rigorously. *DSR* has ≈ 12 (≈ 9) and ≈ 5 (2.4) times fewer unrouted nets than *Std* and *R&R*, respectively for the 10% (20%) new net case. *DSR* was able to route all the new nets 10 and 7.43 times out of 20 runs for the 10% and 20% new net cases, respectively, and is 2.5 and 1.92 times more successful than *R&R* in this regard. *Std* is almost never be able to route all new nets in any single run. Also the percentage of nets that could not be routed by *DSR* over all unsuccessfull runs (runs in which all nets could not be routed) out of 20 runs is ≈ 3 and ≈ 2 times lesser than *R&R* results for the 10% and 20% new net cases, respectively.

As shown in Table 3, the total length of unrouted nets by *Std* and *R&R* are 37 (6.6) and 5 (2) times more than that of *DSR* for the 10% (20%) new net case, respectively. Also the width of unrouted nets for *Std* and *R&R* are 1.6 (2.06) and 1.29 (1.52) times more than that of the unrouted nets of *DSR*. Even though our algorithm obtains good results in the above metrics, it did not increase the average length of successfully routed nets. We obtained the percentage difference %ΔhpBB in the HPs of the pin BB and the routing BB (the former is a lower bound for latter) for each new net as a measure of how effective the different methods are in incrementaly finding minimal-length routes. As seen in Table 3, for all three methods these values are very small. In addition to this, the number of vias used for the new nets by *DSR* is less than those used by *R&R* and close to those used by *Std*. However, we should keep in mind that *DSR* routed many more nets than the other methods (most of them wider nets), used reasonable number of vias and still kept net lengths close to optimal. As also seen in Table 3, *R&R* almost doubled the number of vias for existing nets which are rerouted in order to find a solution for the new nets. However, since *DSR* performs very local modifications on existing nets, the number of vias of existing nets did not change significantly.

As shown in Table 3, while finding a solution for a new net, *DSR* overlaps and tries to re-route many more nets than does *R&R*. Note that this is not the number of existing nets modified in the final solution (which is much smaller) but the number of existing nets which were overlapped and re-routed on a "trial" basis—most of these trials were determined to fail (in order to have good control for minimizing perturbations of existing nets) and retracted. This means that *DSR* has a larger search space than *R&R*. *DSR*'s runtime is thus 3 times more than *R&R*'s, though in absolute terms it is quite fast; e.g., it is able to route on average 93 new nets in

```
Algorithm Ripup-And-Reroute(S)
    Input: The new netlist S will be added ;
    Output: Incremental routing result;
Begin
        Create $OG$;
        for each $n_i \in S$ do begin
            $R\text{-}BB(n_i)$=$BB(n_i)$; /* initialize bounding box
            repeat
                $NUG$=Extract-Non-Uniform-Grid($n_i$, OG);
                result=Route-Without-Overlapping($n_i$, $NUG$);
                if (result==success) then break;
                $P_b$=Get-Best-Path($n_i$); /* use same cost func. as DSR */
                if($P_b \neq null$) then /* $P_b$ can be null when
                                                all candidate paths hit obstacles */
                    Ripup all nets on path $P_b$ and add to netlist $S$;
                    Route net $n_i$ on path $P_b$;
                    break; /* solution found exit from loop */
                endif;
                Expand R-BB($n_i$);
            Until ($R\text{-}BB$ of $n_i$ cannot be extended );
        Endfor
End. /* Ripup-And-Reroute () */
```

Figure 10: *The previously proposed Ripup & Reroute (R&R) incremental routing algorithm.*

Metrics ↓	With 10% new nets			With 20% new nets		
	Std	R&R	DSR	Std	R&R	DSR
# of used vias for new nets	2.87	2.98	2.89	2.87	2.96	2.88
via incr. of modified net (%)	0.0	88.32%	30.25%	0.0	116.68%	23.17%
%ΔhpBB	1.61%	1.49%	1.43%	1.38%	1.31%	1.28%
total HP of F. net	889163	123531	24197	456536	148623	69283
% of Failed net's HP	34.26%	4.76%	0.93%	8.76%	2.81%	1.33%
Avr. # of F. net	8.02	3.43	0.7	13.46	3.82	1.58
Avr. Unrouted nets (%)	17.3%	7.4%	1.6%	14.4%	4.1%	1.7%
Avr. F. net W.	7.2	5.6	3.5	7.3	6.2	4.8
# of re-routings of exisiting nets tried for each new net	0.0	5.6	21.98	0.0	3.24	14.75
Avr. # of modified existing nets for each new net	0.0	2.59	2.04	0.0	1.51	1.25

Table 3: *Comparisons of different metrics for Std, R&R and DSR.*

45.5 secs (see Table 2). We also collected results for global nets, i.e., nets whose lengths are longer than 50% of the HP of the chip; see Table 4. The improvements obtained by our method over *Std* and *R&R* for global nets, for the above discussed metrics, are comparable to the improvements, shown in Table 3, obtained by it for all nets. Hence, our algorithm routes both local and global nets with good efficacy.

As shown in Table 5, we ran *R&R* with different values of ripup-flexibility parameter r to see how the results improve when this flexibility increases. The results do not improve appreciably as r is increased from 5 to 20. Furthermore, the number of unrouted nets by *R&R* was 3.61 (2.16) times more than that of *DSR* for 10% and 20% new nets, respectively, for $r = 20$ for which *R&R*'s solution times are roughly the same as that of *DSR*. This clearly shows that *DSR* is a fundamentally better incremental routing algorithm than *R&R*.

7 Conclusion

A new incremental routing algorithm *DSR* was presented for gridless VLSI circuits with variable width and variable space requirements. The router was tested on several example benchmarks. Our algorithm was shown to produce significantly improved results in terms of the percentage of successfully routed ECO nets (under stringent conditions of using only two routing layers), number of vias required, wire length and the degree of modifications to existing nets when compared to the well-known *Std* and *R&R* incremental routing methods. The *DSR* incremental router thus offers significant advantages in almost all important metrics for incremental routing in VLSI. In future work, we will use a tile-based approach to avoid congested areas when finding solutions for overlapped nets in the DFS process, and thereby improve the speed of our algorithm.

Metrics ↓	With 10% new nets			With 20% new nets		
	Std	R&R	DSR	Std	R&R	DSR
# of used vias for new nets	3.12	3.10	3.10	3.25	3.13	3.14
via incr. of modified net (%)	0.0	93.25%	35.76%	0.0	97.42%	17.21%
%ΔhpBB	0.29%	0.31%	0.34%	0.3%	0.32%	0.35%
Total HP of F. net	157236	23732	2594	327406	24597	7621
% of Failed net's HP	6.06%	0.91%	0.1%	6.29%	0.47%	0.15%
Avr. # of F. net	2.38	0.72	0.21	4.33	0.74	0.34
Avr. Unrouted nets (%)	17.3%	7.4%	1.6%	14.4%	4.1%	1.7%
Avr. F. net W.	6.03	4.1	1.84	6.95	4.13	2.36
# of re-routings of exisiting nets tried for each new net	0.0	5.2	19.52	0.0	2.98	11.99
Avr. # of modified existing nets for each new net	0.0	2.21	1.8	0.0	1.28	1.14

Table 4: *Comparisons of different metrics for Std, R&R and DSR for the nets whose lengths are longer than 50% of the HP of the chip .*

	With 10% new nets				With 20% new nets			
Ripup-Flexibility r	Unrt. Nets	Time (sec.)	Avr. HP of F. net	Avr. F. Net.W.	Unrt. Nets	Time (sec.)	Avr. HP of F. net	Avr. F. Net.W.
5	7.40%	12.08	2030.1	5.6	4.12%	15.61	2249.3	6.2
10	6.55%	18.16	2049.27	5.62	4.01%	20.57	2204.04	6.08
15	5.65%	26.50	1956.72	5.55	3.92%	35.87	2225.75	6.17
20	5.75%	34.75	2044.46	5.57	3.73%	43.62	2196.84	6.05

Table 5: *R&R results for different values of ripup flexibility parameter r.*

References

[1] J, Cong, C. -K. Koh, and P. Madden, "Performance optimization of vlsi interconnect layout" Integration, the VLSI Journal, vol 21, pp. 1-94, 1996.

[2] J. Cong, J. Fang and K. Khoo. "An Implicit Connection Graph Maze Routing Algorithm for ECO Routing". *ICCAD*'99, pp. 12-18.

[3] Kawamura etal."Touch and Cross Router". *ICCAD'90*, pp. 56-61.

[4] Y. -L. Lin, Y. -C. Hsu and F. -S. Tsai, "Silk: A Simulated Evolution Router". *IEEE Trans. CAD*, 8(10), 1989.

[5] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-Based Performance-Driven Router for FPGAs". *ACM FPGA Symp.*, pp. 111-117, 1997.

[6] J. M. Emmert and D. Bhatia, "Incremental Routing in FPGA's". *Proc. IEEE Int. ASIC Conference and Exhibit*, 1998.

[7] J. Cong and M. Sarrafzadeh. "Incremental Physical Design". *ISPD* , April 2000, pp. 84-92.

[8] J. D. Carothers and D. Li. "The MCG Autorouter for Multichip Modules". *IEEE Trans. on Circuits and Systems*, 45, 1999.

[9] Joseph Jaja and S. Alice Wu. "On Routing Two-Terminal Net in the Presence of Obstacle". *IEEE Trans. CAD* pp. 563-570, 8(5), 1989.

[10] Y. Wu and P. Widmayer and M. Schlag and C. Wong, "Rectilinear Shortest Paths and Minimum Spanning Trees in The Presence of Rectilinear Obstacles". *IEEE Trans. Computers* C-36(1), 1987.

[11] S. Zheng and J. S. Lim and S. Iyengar, "Finding Obstacle-Avoiding Shortest Paths Using Implicit Connection Graphs". *IEEE Trans. CAD*, 15(1), 1996.

[12] S. Dutt, V. Verma and H. Arslan, "A Search-Based Bump-and-Refit Approach to Incremental Routing for ECO Applications in FPGAs", *ACM (TODAES)*, 7(4), pp. 664-693, 2002.

[13] S. Dutt, V. Shanmugavel and S. Trimberger, "Efficient Incremental Rerouting for Fault Reconf. in FPGAs", *ICCAD*, pp. 173-176, 1999.

[14] K. -Y. Khoo and J. Cong,"An Efficient Multilayer MCM Router Based on Four-Via Routing", *ACM-DAC*, 1993.

[15] J. Cong, J. Fang and Y. Zhang, "Multilevel Approach to Full-chip Gridless Routing", *ICCAD*, 2001.