A Search-Based Bump-and-Refit Approach to Incremental Routing for ECO Applications in FPGAs

SHANTANU DUTT, VINAY VERMA, and HASAN ARSLAN
University of Illinois at Chicago

Incremental physical CAD is encountered frequently in the so-called engineering change order (ECO) process in which design changes are made typically late in the design process in order to correct logical and/or technological problems in the circuit. Incremental routing is a significant part of an incremental physical design methodology. Typically after an ECO process, a small portion of the circuit netlist is changed, and in order to capitalize on the enormous resources and time already spent on routing the circuit it is desirable to reroute only the ECO-affected portion of the circuit, while minimizing any routing changes in the much larger unaffected part. Incremental rerouting also needs to be fast and to effectively use available routing resources. In this article, we develop a complete incremental routing methodology for FPGAs using a novel approach called bump and refit (B&R). The basic B&R idea (which was originally proposed in Dutt et al. [1999] in the much simpler context of extending some nets by a segment for the purpose of fault tolerance) in our algorithms is to rearrange some portions of some existing nets on other tracks within their current channels in order to find valid routings for the new/modified nets without requiring any extra routing resources and with little effect on the electrical properties of existing nets. Here we significantly extend the B&R concept to global and detailed incremental routing for FPGAs with complex switchboxes (SBox's) such as those in Lucent's ORCA and Xilinx's Virtex series. We introduce new concepts such as a B&R cost in global routing and the optimal subnet set to relocate for each bumped net (determined using an efficient dynamic programming formulation). We developed optimal and near-optimal algorithms (called `Subsec_B&R` and `Subnet_B&R`, respectively) to find incremental routing solutions using the B&R paradigm in complex FPGAs (e.g., Lucent's ORCA FPGA) with i-to-j SBox's, as well as an optimal version `Fullnet_B&R` for the VPR architecture from the University of Toronto using the simpler i-to-i SBox's. We compared our algorithms (simply called `B&R` when no distinction needs to be made between our versions) to two recent incremental routing techniques, `Standard (Std)` and `Rip-up&Reroute (R&R)`, and to Lucent's `A_PAR` routing tool and the University of Toronto's `VPR` router used in complete rerouting modes. Experimental results for the ORCA show that B&R is 10 to 20 times faster than complete rerouting using `A_PAR`, and that B&R is also nearly 27% faster and yields new nets with nearly 10% smaller lengths compared to previous incremental routers. Furthermore, B&R routers do not change either the lengths or topologies of existing nets, a significant advantage in ECO applications, in contrast to `R&R` which increases the length of ripped-up nets by an average of 8.75 to 13.6%. Experimental results for the VPR architecture are dominated by the significantly larger (in many cases, orders of magnitude more) number of nets left

This work was funded in part by DARPA Contract #F33615-98-C-1318 and in part by a grant from Xilinx Corp.
Authors' address: Dept. of Electrical and Computer Engineering, University of Illinois at Chicago, 1020 Science and Engineering Offices, 851 S. Morgan Street, Chicago, IL 60607-7053; email: dutt@ece.uic.edu.

unrouted by Std and R&R compared to B&R, which highlights the much greater efficacy of B&R-based incremental routing. However, B&R is significantly slower than the other two incremental routers, although on an absolute scale it is quite fast for two of four cases we simulated; in one case, it is about 25 times faster than VPR used in the full rerouting mode. The relative slowness of B&R for the VPR architecture arises from the fact that we used i-to-i SBox's which forces each net to be routed on the same track, thus causing significantly more bumpings and searches for rearranged solutions compared to i-to-j SBox's where a net can be routed on different interconnected tracks to minimize the amount of bumpings (as we did for the ORCA). Since modern FPGAs generally have the latter type of SBox's, B&R would be fast as well as very effective on them.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids; J.6.0 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

General Terms: Algorithms, Design

Additional Key Words and Phrases: Bump-and-refit (B&R) paradigm, bumping cost, detailed routing, dynamic programming, ECO (engineering change order), field programmable gate arrays, global routing, incremental routing, switchbox

1. INTRODUCTION

An FPGA is a general-purpose, programmable logic device that is customized in the package by the end user. It consists of a large number of programmable logic blocks (PLBs) or cells and programmable routing that allows the logic blocks to be connected to form a larger circuit; see Figure 1. The logic is implemented by electronically programming the interconnects, typically by the user instead of the manufacturer.

An SRAM-programmable FPGA is programmed by loading configuration memory cells from an external source. The configuration memory cells control the logic and interconnect that perform the application function of the FPGA. Such FPGAs are reprogrammable, and can be used to implement different functions at different times. Such reprogrammability also allows much flexibility for ECO applications late in the design process.

Due to the complexity of current generation VLSI systems and the iterative design process, it frequently happens that late design changes need to be incorporated at one or more levels of the design process which results in the so-called *engineering change order* or *ECO*. To tackle such design changes in a cost-effective and seamless manner, very efficient incremental design algorithms and methodologies are needed. Incremental routing is an integral part of any incremental physical design methodology, and almost all ECO scenarios require incremental rerouting. For example, the netlist might be incrementally updated by the designer or a synthesis tool to meet, say, low-power requirements, or some nets might be determined to have timing/noise violation after parasitic extraction and timing analysis. In such cases, redoing the routing for all the nets is too time consuming and can play havoc with time-to-market requirements. Moreover, an entirely different layout may completely invalidate the detailed timing results, which is undesirable. It is thus desirable to reroute only the ECO-affected portion of the circuit, while minimizing any routing changes in the much larger unaffected part of the circuit. Incremental rerouting also needs to be fast and to effectively use available routing resources

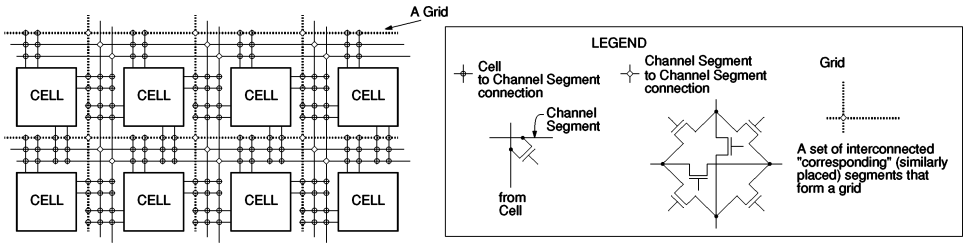

Fig. 1. FPGA generic channel wiring architecture with segment length spanning one cell/PLB. The set of dashed/dotted segments forms a *grid*.

in order to avoid time- and area-expensive processes such as re-floorplanning, re-placement, and channel expansion to complete the required routings.

In this article we present incremental routing algorithms for complex FPGAs that use a novel bump-and-refit (B&R) approach. This approach was initially proposed in Dutt et al. [1999] in the much simpler context of extending some nets by a segment (for the purpose of fault tolerance) for FPGAs with simple i-to-i switchboxes (SBox's). Here we significantly develop this concept further to design a complete incremental routing flow for complex FPGAs. We introduce new concepts such as bumping costs during global routing, and optimal bumping subsets of nets to realize an efficient incremental routing technique for ECO applications.

The goals of our work are to develop incremental routing algorithms that (1) are orders of magnitude faster than complete rerouting; (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 fallback strategies); and (3) complete the routing without significantly changing electrical properties (e.g., power, delay) of existing nets (this will keep the parasitic extraction data and timing/power analysis for the unaffected portion of the circuit valid). Experimental results show that the new B&R incremental routers have significant advantages over existing incremental routing methods with respect to the above metrics. We also prove optimality of parts of our algorithms.

The rest of the article is organized as follows. In Section 2, we discuss previous research in incremental routing. Section 3 covers all aspects of the new B&R incremental routing method. Experimental results comparing two versions of the B&R incremental router to the type of incremental routers proposed in Emmert and Bhatia [1998] and Cong and Sarrafzadeh [2000] that we have implemented are given in Section 4. We conclude in Section 5.

2. PRIOR WORK ON INCREMENTAL REROUTING

This is a new area with relatively few papers that have tackled this problem. One of these is the incremental rerouting technique developed by Emmert and Bhatia [1998]. In their work the nets connected to faulty/displaced logic blocks (PLBs) are partially ripped up and rerouted. Graph building between the pin of the starting channel (SC) and the pin of the target channel (TC) is attempted. If this is successful, the path with the minimum cost is selected. If the router is not successful, the window size for the router is increased by one unit. Unlike

Dutt et al. [1999], Cong and Sarrafzadeh [2000], and our current work, they do not perturb or move the already routed nets. We term this incremental routing approach as `Standard` in the rest of the article.

Cong and Sarrafzadeh [2000] present a rip-up and reroute approach to incremental physical design. The first goal of their incremental routing is to route the new nets without removing any existing nets. When some nets cannot be routed, a rip-up and reroute procedure is used to free routing resources and redo routing for the newly added as well as ripped-up nets. If rip-up and reroute fails to route all the nets, the floorplan and placement of the design is updated to add more routing resources. Their technique of rip-up and reroute is in contrast to our B&R approach wherein we perturb existing nets only within their current channels, rather than rerouting them. Because the nets are still routed in their original channels, neither their topologies nor their lengths change. Thus most properties of existing nets are preserved which is key to an effective incremental design process.

Cong et al. [1999] presented efficient techniques for obtaining a nonuniform 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 chips, but it is orthogonal to the problem of developing incremental routers, which is the topic of this article.

Another recent work is that of Dutt et al. [1999] in which an incremental rerouting algorithm was developed for fault reconfiguration in FPGAs. We discuss this work in detail here as it is closely related to and sets the background for the current work. In Dutt et al. [1999] segmented FPGAs that use i-to-i connection SBox's are considered that; that is, in such a routing architecture, a net is routed on only one track throughout its entire path. It uses the concept of node cover as in Hanchek and Dutt [1998], to cover cell (PLB) faults but is different from that paper in the manner in which net extensions (also called *cover segment (CS)* insertions) are made for the purpose of fault reconfiguration. It makes CS insertions only specific to faults wherever and whenever required in a dynamic manner (Hanchek and Dutt [1998] use a static method to provide CSs to cover all possible requirements for the given fault pattern, one fault per row). In Figure 2(a), net n_1 is a CS net, a net that needs to be extended by one segment (the CS) in order to connect to the PLB (called the *cover* cell) that replaces the original PLB (for the purpose of fault reconfiguration). For each CS, if the required track segment is vacant, the insertion is accomplished by including this segment as part of the corresponding net. However, if the required wire segment is occupied by another net, then the CS insertion will cause a displacement or "bumping" of this net.

As shown in Figure 2(a), the net occupying the required track segment is termed the *occupying net (O-net)*. The CS-net has to be extended by one segment towards the direction of the cover cell, and this segment is currently occupied by the O-net n_2. Thus the O-net needs to be moved out of its current track. Let a *transition* be defined as the movement of net n_i on a track T_j to another track T_k, and denoted by $n_i^{T_j \rightarrow T_k}$ (we also use $n_i^{T_j}$ to denote a net n_i on track T_j). This transition may result in net n_i bumping into one or more nets on track T_k. These nets will have to move out of their current track T_k, giving rise to a

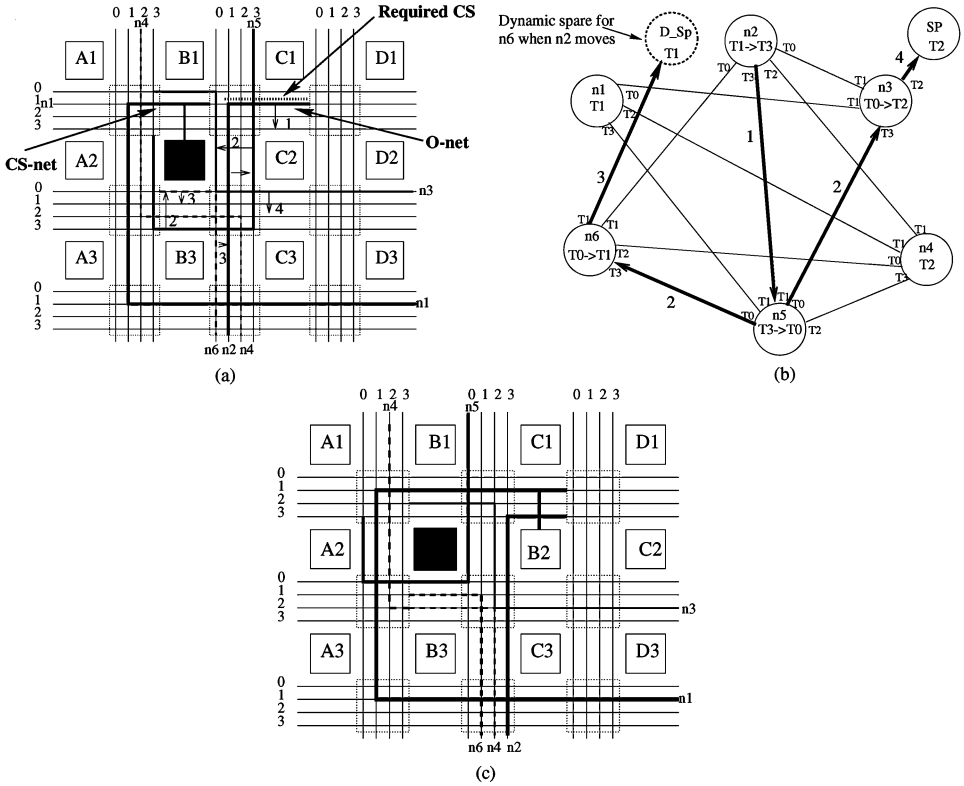

Fig. 2. (a) Presence of occupying net (O-net) n_2 prevents a straightforward insertion of a CS. Thus the O-net must be moved to another track, possibly bumping other nets. (b) Overlap graph representation for the given routing. The track number at an endpoint of an edge indicates the track the corresponding net would move to if this edge is traversed. (c) Final routing of the circuit.

transition for each of them. This transition sequence is shown in Figure 2(b) by dark arcs, where net n_2 initiates a set of transitions that finally terminate in "spare" nodes, which are vacant segments of appropriate total lengths in which a bumped net can move in without bumping any other net. The set of transitions takes on a directed-acyclic graph (DAG) structure, termed a *transition DAG (T-DAG)*, with the spares forming the leaf nodes. The CS insertion is successful if a T-DAG rooted at the corresponding O-net can be found whose leaves are spare nodes; such a T-DAG is termed a *converging T-DAG*.

The concept of an *overlap graph* (OG), which is a graph representation of the circuit routing on the FPGA, is introduced in Dutt et al. [1999] as an aid to finding a converging T-DAG solution. The OG is an undirected graph with the circuit nets represented by the nodes of the graph. There exists an edge between n_i and n_j in the OG if and only if nets n_i and n_j share a channel[1] in the FPGA. Figure 2(b) shows nets n_2 and n_6 having an edge between them in the OG since in Figure 2(a) they are routed through a common vertical channel to the right of cell B3.

[1]A *channel* is the set of all track segments between two adjacent SBox's of the FPGA.

The OG can be used to determine if the required T-DAG exists. Because the OG represents the circuit routing in the FPGA, a T-DAG is a DAG embedded in the OG (the undirected edges of the OG become directed arcs in the direction of the transitions; see Figure 2(b)). Thus a converging T-DAG rooted at an O-net can be determined by performing a search for a T-DAG on the OG.

This process is illustrated in Figure 2 for a small circuit and for a single new net n_1. The corresponding O-net n_2 transits from T_1 to T_3 and bumps into n_5. The movement of n_2 from T_1 creates a "dynamic" spare node (labeled D_Sp in Figure 2(b)) for net n_6, which information is added to the OG. The bumped net n_5 then transits from T_3 to T_0 where it bumps n_6 and n_3. n_6 then transits to the above dynamically created spare on T_1, while n_3 transits to its spare track segments on T_2. Thus a converging T-DAG is determined in the OG. The transition arcs are shown dark in Figure 2(b) and numbered chronologically in the order in which they are traversed in the search process. Figure 2(c) shows the final routing of the FPGA after the bumping sequence converges.

A depth-first search algorithm for finding a converging T-DAG in the OG was developed in Dutt et al. [1999]. This algorithm is optimal in terms of finding a converging T-DAG if one exists. An extended version of this algorithm for more complex FPGAs and for ECO applications is presented later in Figure 11.

Although an existing converging T-DAG will ultimately be found by the depth-first algorithm of Dutt et al. [1999], it will be time-efficient if some suitable "cost" measure can be used to determine which transitions are more likely to be successful so that fewer T-DAGs are searched and backtracked. A good cost measure will consider both the "magnitude" of bumpings (total length of bumped nets) and the likelihood of convergence of these bumpings.

Two transition cost (TC) (equivalently, *bumping cost*) measures evaluated are as follows.

1. $sum(n_i^{T_j \to T_k}) = \sum_{n_j \in adj^{T_k}(n_i)} l(n_j)$, where $adj^{T_k}(n_i)$ are the neighbors of n_i in the OG that are on track T_k, and $l(n_j)$ is the total length of n_j in terms of the track segments (each of length 1) that it occupies. This cost estimate is reasonable, but only considers the bumping magnitude. For example, according to it, it is equally costly to bump a net of length 9 as it is to bump three nets each of length 3. However, the latter case has a higher likelihood of convergence since there is greater flexibility in moving three bumped nets than a single net of the same total length. This leads to the next cost function.
2. $sqrt(n_i^{T_j \to T_k}) = [\sum_{n_j \in adj^{T_k}(n_i)} l(n_j)]/\sqrt{|adj^{T_k}(n_i)|}$.

Using such TC functions to guide the search results in computation time reduction by an order of magnitude compared to a "blind" depth-first search.

The algorithm of Dutt et al. [1999] is efficient in terms of both track usage and time. It shows considerable improvement over the static method [Hanchek and Dutt 1998] in track overheads for tolerating a single PLB fault in each FPGA row. The static method has a track overhead of 42%, whereas the depth-first algorithm of Dutt et al. [1999] has an average overhead of only 12.8%, a 70% improvement.

In Dutt et al. [1999] the affected nets (nets connected to displaced PLBs) are not rerouted, but only net extensions (CS insertions) are made. So Dutt et al. [1999] in its present framework cannot be used for full-scale ECO routing. We use a B&R approach of finding new track assignments for existing nets in order to make room for new nets that is similar to that of Dutt et al. [1999]. We incorporate the B&R approach within the context of a complete routing system in which we perform full-fledged incremental routing (as opposed to extending some nets by one segment as in Dutt et al. [1999]). This includes performing global as well as detailed routing taking the potential *bumping cost* into account (besides other cost measures). Furthermore, here we also extend the B&R approach to FPGAs with complex switching and routing capabilities such as those available in commercial FPGAs such as Xilinx's Virtex and Lucent's ORCA. The SBox's in these FPGAs allow routing a net on different tracks by interconnecting a segment on track i to another segment on track j. This adds another dimension to the B&R approach (viz., when a net is bumped in one portion, should only that portion be moved to a different track or should more than that portion be bumped (possibly the entire net)?). We have developed a dynamic programming algorithm to optimize metrics such as the probability of successful B&R (as measured by the degree of bumping or bumping cost), and SBox and track resource usage to determine how to bump different subnets of a bumped net. All these aspects of our B&R incremental router are discussed in the next section.

3. A NEW INCREMENTAL ROUTING ALGORITHM

We have partitioned our incremental rerouting problem into a two-stage hierarchy. We first perform global routing taking possible bumping cost into account. Detailed routing is performed next possibly with bumping existing nets if a set of unoccupied interconnectable track segments is not found to accommodate the net being routed. If existing nets are bumped by the detailed router, the bump-and-refit algorithm is used to refit the bumped nets on other track segments in a recursive manner. The incremental B&R routing flow that we have developed is shown in the flowchart in Figure 3 (we have not developed the incremental channel expansion/placement/floorplanning phase shown in the rightmost box of Figure 3). As shown in the flowchart, both the global and detailed routing phases consider costs that control both net length (this is represented by the *base_cost* portion of the total edge cost in the global or detailed routing graph) and potential bumping cost to obtain a min-cost routing (e.g., the global router will prefer to route along channels where the bumping cost, if any, will be potentially minimal as long as a minimal-length route is obtained).

Before going into the details of our incremental router we define a few terms. We define a *subnet* as a maximal portion of a net that spans at least one horizontal or vertical channel and no part of which can be independently moved to another track without disconnecting it from the rest of the subnet; see Figure 4. A subnet lies on a single track. We denote a subnet s_j of net n_i by $n_i.s_j$. We define a *subsection* of a net as a set of one or more interconnected subnets of a net.

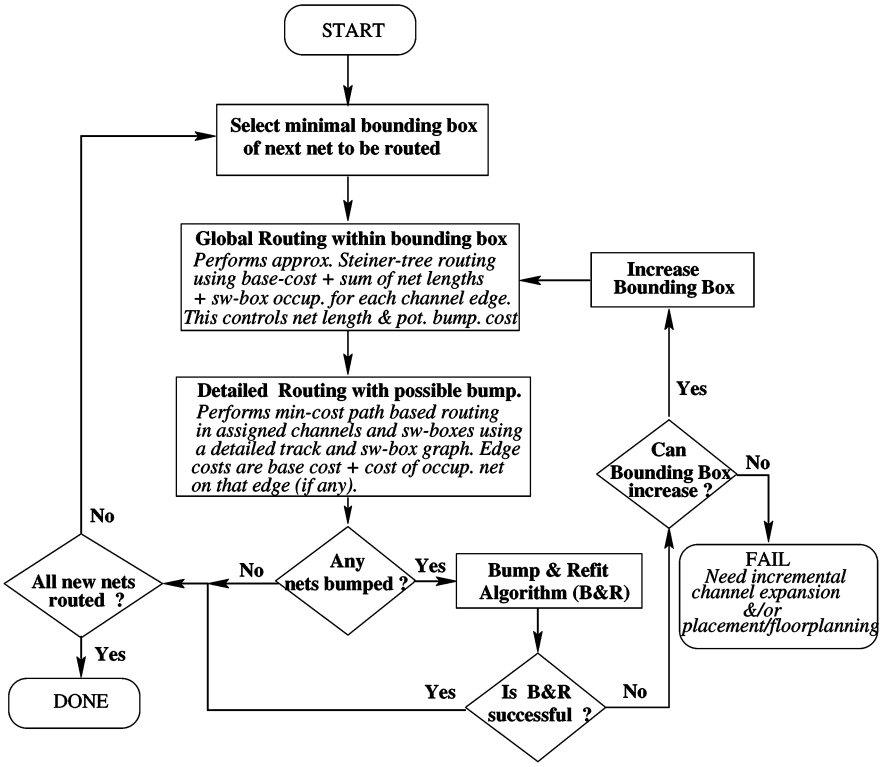

Fig. 3. An incremental routing flow incorporating B&R.

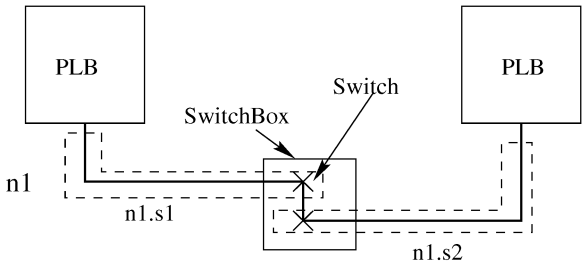

Fig. 4. s_1 and s_2 are subnets of net n_1.

3.1 Global Routing

In global routing we try to optimize the overall wire length of the net to be routed, congestion of the channels and possible bumping cost that may be incurred by the detailed router. The global router assigns channels (or identically SBox's) to the net to be routed. The global routing graph *GR-Graph* is a weighted connection graph of SBox's of the FPGA. Each node in the graph corresponds to a SBox and there exists an edge between two nodes in the graph if the corresponding SBox's are adjacent in the layout; that is, these edges represent

```
Algorithm GRoute(n_i)   /* Steiner-tree approx. global routing using repeated application of Dijkstra's shortest path based on a "distance" metric or cost that is a weighted sum of: (1) net length, (2) channel congestion, and (3) a measure of bumping probability of existing nets by the detailed router */
Begin
    GTree = ∅;
    Sort pins of n_i by Manhattan distance to center of BBox;
    Let the sorted order be (p_1, p_2, ..., p_k);
    /* First obtain a route between p_1 and p_2 */
    Obtain bounding box, BBox(p_1,p_2) ;
    GTree = GTree∪ MinCostPath(p_1, p_2) ; /* Dijkstra's shortest path algo in BBox (p_1,p_2) */
    Mark nodes on path as sink (= s) ;
    q ← center(p_1,p_2) ; /* q is the new center of BBox*/
    for p ← p_3 to p_k
        Obtain bounding box,BBox(p,q); /* from previous center to the current pin */
        GTree = GTree∪ MinCostPath(p,s); /*from p to one of the sinks(=s) */
        Mark all nodes on path as sink (= s) ;
        q ← center(p,q) ;
    endfor
return(GTree); End
```

Fig. 5. Steiner tree heuristic for global routing.

channels between SBox's. The weight/cost function of an edge in the graph is discussed in Section. 3.1. For multipin nets, global routing amounts to finding the approximate Steiner tree connecting these pins. We solve this problem by repeated application of Dijkstra's shortest path algorithm [Cormen et al. 1990] in a bounding box of nodes. A formal description of this heuristic is given in Figure 5.

Note that our main goal is to develop an incremental detailed router using a B&R approach. We, however, need a global router that can also take possible bumping cost into account when making channel assignments. We thus have used a simple global routing strategy with a primary goal of providing a bumping cost factor in the global routing output, so that the detailed router incurs minimal bumping cost. This in turn means that the B&R algorithm, if invoked by the detailed router, will have a high likelihood of finding a solution.

Cost Functions for Global Routing. In its most basic (non-B&R) form, the cost of an edge e_i between two nodes in the GR-Graph is the weighted sum of two metrics, a congestion measure of the channel represented by e_i and a constant *base_cost* that is used to control the length of the net. The congestion in a channel is captured by the resource usage in the SBox of that channel, which is basically the number of switches used in the SBox.

We have used two cost functions. $gcost1$ is used for the the non-B&R routing schemes, and for an edge e_i in the GR-Graph it is given by

$$gcost1(e_i) = \alpha \cdot base\_cost + \beta \cdot s\_box(e_i), \tag{1}$$

where α and β are the weighting factors for the net length and channel congestion metrics, respectively, and $s\_box(e_i)$ is the number of switches used in the SBox connected to the right or bottom of the channel represented by e_i.

The routing cost $gcost2$ for a B&R-based global router needs an extra cost metric to estimate possible net bumping cost that may be incurred later by the detailed router:

$$gcost2(e_i) = \alpha \cdot base\_cost + \beta \cdot s\_box(e_i) + \gamma \cdot \sum_{n_j \in e_i} l(n_j). \quad (2)$$

The third term $\sum_{n_j \in e_i} l(n_j)$ in Equation 2 is the total length of nets in the SBox represented by e_i ($l(n_j)$ is the length of net n_j) and captures the global bumping cost; for example, it is easier in general to find converging T-DAGs by bumping shorter nets than longer ones.

3.2 Detailed Routing

After the global router assigns channels (or SBox's) to the net to be routed, detailed routing is performed. The objective of detailed routing is to make a feasible assignment of tracks and switches for the net with minimum utilization of routing resources (tracks in channels and switches in SBox's).

The general SBox model that we use is one in which smaller switches within a SBox are able to be shared between various $T_i \rightarrow T_j$ interconnections (as opposed to being dedicated to specific interconnections) with only one connection being able to use a switch at any time. Thus a switch resource cost (= the number of switches used) is incurred whenever a specific $T_i \rightarrow T_j$ connection is made via a SBox. Just as in track segment assignments, it is also possible to bump into other nets if a particular $T_i \rightarrow T_j$ connection for a net n_k uses a switch currently occupied by another net n_l; in such cases besides the switch resource cost, a bumping cost (see Equation (3) below) is also incurred. This bumping of n_l also needs to be resolved in a manner similar to the bumping of a net from a track segment; the integrated bump-and-refit for both types of net bumping is solved by Algorithm `B&R` given later in Figure 11. The model can also be extended to SBox's with some or all dedicated switches used to make only specific $T_i \rightarrow T_j$ connections by not having any resource cost associated with the use of such switches. This SBox model is general enough to capture SBox's found in commercial FPGAs such as ORCA, Virtex, and Virtex-II; parameters such as switch resource cost and bumping cost within a SBox will need to be appropriately instantiated to apply to a specific FPGA.

Figure 6 shows a specific type of SBox that is similar to one used in the ORCA, and which follows the general SBox model. The switches in the ORCA are not dedicated and can be used to connect in either the vertical or horizontal direction. Hence there is a cost associated with their usage. A SBox is modeled as a connection graph called the detailed routing graph *DR-Graph*. The switches of a SBox are nodes in the DR-Graph. Two switches that can be directly connected electrically have an edge between them. A track is also modeled as a node (called a segment node) in the DR-Graph that has an edge to all switches of the SBox on that track. A segment node is used to enter and leave the SBox. Also the pins

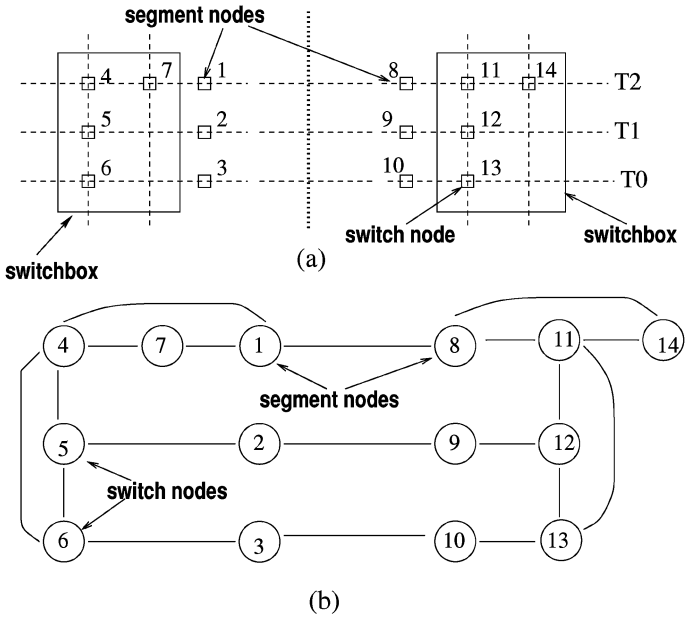

Fig. 6. (a) Internals of a SBox showing switches. The switch box is similar to the one in the ORCA FPGA. (b) Connection graph of switches (DR-Graph) used in detailed routing.

of a PLB have equivalent pin nodes in the DR-Graph. In Figure 6(a) switch 1 is a segment node of the left SBox on track T_2. It has an edge to switches 4 and 7 on track T_2 in the same SBox. If the two SBox's are connected in the GR-Graph then the corresponding pair of segment nodes on the same track is connected in the DR-Graph. In Figures 6(a) and (b), segment nodes 1 and 8 correspond to the same track T_2 and are adjacent in the DR-Graph. In Figure 6(b), node 8 has edges to both switch nodes 11 and 14, because a net on the track segment corresponding to node 8 can connect to the track segment to its right via switch node 14 and bypassing switch node 11; thus another net can use node 11 to make a vertical connection.

In Algorithm DRoute, detailed routing is performed in the order in which the corresponding branches were created in the global routing tree *GTree*; see Figure 7.

Cost Function for Detailed Routing. The cost of a switch node sw_i in a DR-Graph for detailed routing is given by

$$dcost(sw_i) = \alpha \cdot sw\_cost + \beta \cdot l(sw_i), \tag{3}$$

where α and β are weighting factors, sw_cost is a constant cost of using a switch node in the SBox, and $l(sw_i)$ is the length of the net, if any, that is currently using switch sw_i; it thus represents the bumping cost for any new net that needs to use an occupied switch.

Figure 8(a) shows the SBox of the Xilinx XC4000 FPGA which allows only i-to-i connections. As shown here, there is no sharing of the atomic switch

```
Algorithm DRoute(n_i)
Begin
1    B = ∅;  P = ∅; /* B is the set of bumped subnets, P is the detailed route to be
     determined. */
2    for each branch Br of GTree /* GTree is the channel Steiner tree returned by
     GRoute */
3        if( Br is the first branch of GTree)
4            P = P ∪ MinCostPath(p_i, p_j); /*p_i, p_j are pin nodes on the first branch of
             GTree; MinCostPath is Dijkstra's shortest path algorithm.*/
5            if MinCostPath(p_i, p_j) bumps existing nets
6                Get bumped subnets in set B;
7        endif
8        else begin
9            Let Br = (p_k, S), where p_k is a pin node and S a SBox determined as the
             Steiner point for Br by GRoute;
10           Mark all switches in SBox S used by the detailed route created so far for n_i
             as s /* potential sinks for connection to p_k */
11           P = P ∪ MinCostPath(p_k, s); /*min. cost path from pin node p_k to any
             switch node s which is marked.*/
12           if MinCostPath bumps existing nets then
13               Insert bumped subnets in set B;
14       endelse
15   endfor
16   for each subnet s_i ∈ B /* the bumped subnets */
17       flag = B&R(s_i) ; /* use the B&R algorithm described in Figure 11 */
18       if (flag == failure) then return(flag, ∅);
19   endfor
20   return(success, P);
End
```

Fig. 7. Algorithm for detailed routing with possible bumping of existing nets.

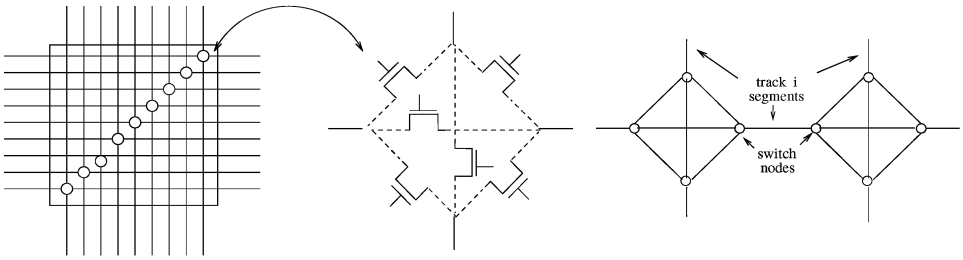

Fig. 8. (a) Internals of a Xilinx XC4000 FPGA SBox showing switches that allow only i-to-i interconnections. (b) Part of the connection graph of switches (DR-Graph) representing two adjacent SBox's for a single track i used in detailed routing (for t tracks, the entire graph consists of t disjoint copies of this single-track graph). Note that switches here are not shared among different possible interconnections among track segments and thus the concept of switch cost does not arise for this case in the detailed routing phase. Note also that since the XC4000 SBox only allows i-to-i connection, detailed routing in this case simply reduces to finding the track t along the global path of a net n_i that is of the least cost among all t tracks according to some suitable cost metric (e.g., one with the least occupancy or least sum of lengths of occupying nets, etc.); algorithm DRoute (Figure 7) is not needed for such a simple routing architecture.

resources (the pass transistors) among different i-to-i connections such as North–East, East–West, and so on, and thus the *sw_cost* portion of the cost in *dcost* is zero for such a SBox. Equation (3) is a general enough expression that can capture the routing cost through different SBox architectures with different degrees of sharing of atomic switches among different interconnections that are possible through the SBox.

3.3 Detailed Routing with Bump and Refit

Figure 7 describes our detailed router DRoute that incorporates B&R when a nonbumping path does not exist for the current net being routed. If β is made much larger than α in Equation 3, it is clear that DRoute would choose a nonbumping solution if one exists in the channels allocated by the global router.[2] The horizontal channel in the ith routing row is denoted by H_i, and a vertical channel in the jth routing column is denoted by V_j. Sometimes the routing resources are not available in those channels to complete a valid route; see Figure 9(b). In Figure 9(a), n_1 needs to be to be rerouted and connected to PLB C_1. The global router allocates channel H_1 for the reroute. In Figure 9(b), net n_4 occupies track T_2 in channel H_1 from V_2 to V_3. Net n_2 occupies track T_1 in channels H_1 from V_1 to V_3, and net n_3 occupies track T_0 in channel H_1 from V_0 to V_1. There are no routing resources available for net n_1 to connect from PLB A_1 to C_1. Hence the detailed router fails to find a valid route. However, if we bump n_4 to track T_0, track T_2 in channel H_1 is vacated for n_1, and a valid route from PLB A_1 to PLB C_1 is created for net n_1; see Figure 9(c).

A *Contiguity Graph (CG)* of a net is the connection graph of subnets. Each subnet is a node in the graph. The subnets that are interconnected in the net's routing have an edge between them in CG. The concept of an *overlap graph* is similar to the definition in Section 2 [Dutt et al. 1999] with a subnet being a node in the graph instead of a net.

Figure 10 shows an example of circuit routing and how the *OG* and *CG* are created. Subnet $n_1.s_1$ is connected to subnet $n_1.s_2$ (see Figure 10(a)), and hence their corresponding nodes have an edge in the *CG*; see Figure 10(b). Subnets $n_1.s_1$ and $n_2.s_1$ share a channel in the FPGA (see Figure 10(a)), and hence their corresponding nodes have an edge in the *OG*; see Figure 10(c) .

3.4 The B&R Algorithm

The detailed router may not always find a nonbumping path for the net that needs to be rerouted. We call such a net an *R-net*. The R-net bumps out the nets occupying routing resources it needs; these nets are the *O-nets*. The subnets of the O-nets that occupy the routing resources of the R-net are called *O-subnets*. The O-subnets have to make transitions to different tracks in order to realize a valid detailed routing. The formal description of the algorithm for accomplishing this is given in Figure 11.

[2]Although this is not necessarily desirable, because longer switch routes within SBox's (this, however, does not contribute significantly to any net length increase, but increases resource usage and net delays) may be chosen to avoid bumping, and a bumping solution may be able to rearrange the nets so that each uses near-minimum switch routes.

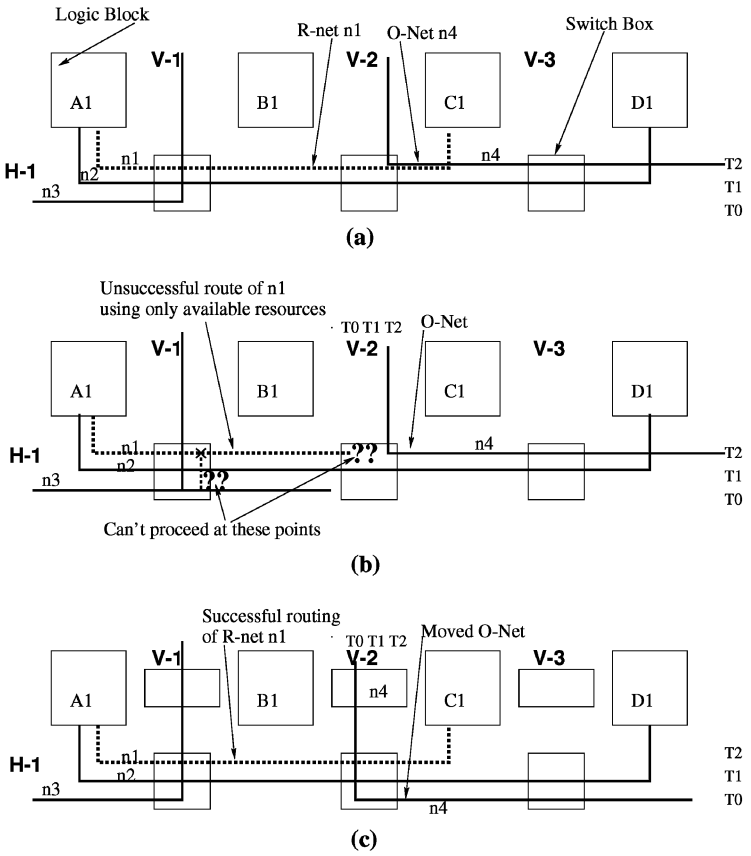

Fig. 9. (a) A new/modified net n_1 needs to be routed to connect A_1 and C_1. (b) A non-B&R detailed router is unable to route through the channels allocated by a global router. (c) A B&R solution to the routing of n_1 involves bumping net n_4 to track T_0, thereby allowing net n_1 to be routed on T_2.

We have developed two strategies for performing B&R: `Subnet_B&R` and `Subsec_B&R`. In the `Subnet_B&R` method we bump only the O-subnets of the O-net. The cost of bumping a subnet s from track T_i to T_j is given by Equation (4):

$$bumpcost(s^{T_i \to T_j}) = \text{length of subnet on } T_j \text{ bumped by } s. \tag{4}$$

In Equation (4), since s is a subnet, it may bump at most one subnet of another net. This bumping cost is similar in concept to the one described in Section 2 [Dutt et al. 1999]. The subnets contiguous to O-subnets remain in their original tracks and maintain electrical connections to the O-subnets via the intervening SBox's. This may require some extra switch usage of the SBox's. If the required switches are not available then the nets occupying them are bumped out.

The `Subsec_B&R` approach was designed to reduce both the bumping cost as well as the switch usage cost in the SBox to maintain electrical connection between contiguous subnets of the O-net. In this method we compute the optimal subsection of the O-net that should be bumped in order to minimize a weighted sum of the above two costs. The optimal subsection includes the O-subnets and

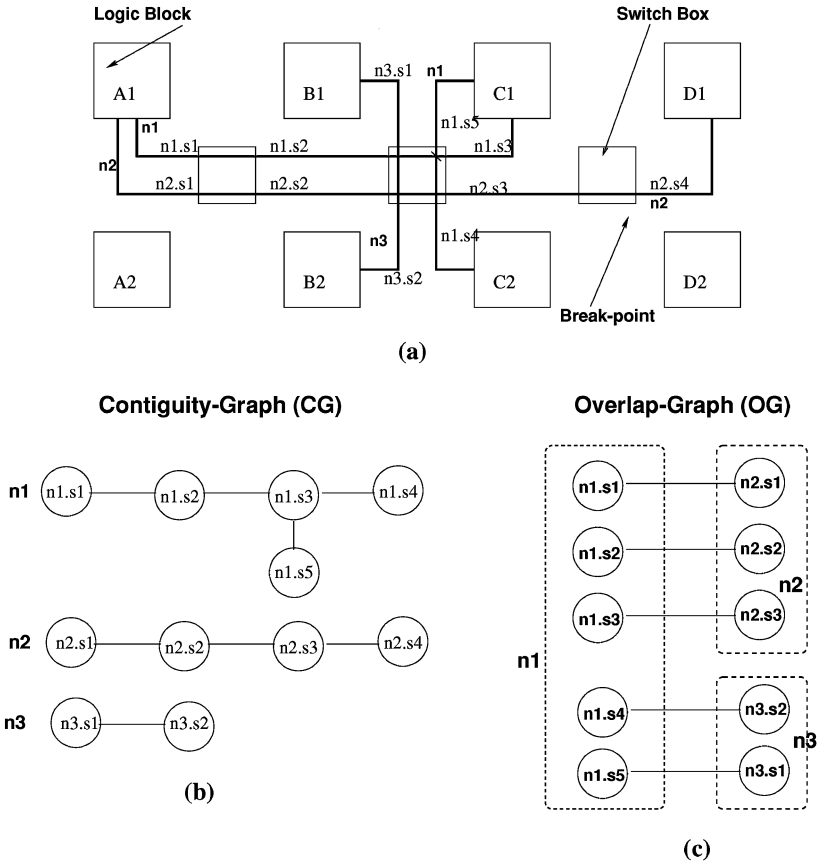

Fig. 10. (a) Routing of nets and their subnets. The corresponding (b) contiguity graph and (c) overlap graph.

possibly some other subnets of the O-net. To compute the optimal subsection, we recursively compute the minimum transition cost of subnets to be perturbed. The transition cost of subnet s_i to any track T_k is the sum of its bumping cost and the cost to connect it to all subnets adjacent to it in its CG.

In Figure 12, subnet x^{T_i} is the *O-subnet* of net n_r, and subnets y^{T_p} and z^{T_l} are adjacent to x in the *CG*. The computation of the minimum transition cost $dynamcost(x^{T_i \to T_j})$ of moving a generic subnet x from track T_i to track T_j is given by

$$dynamcost(x^{T_i \to T_j}) = \min_{\forall T_q}[SBox^{xy}(T_j \to T_q) + dynamcost^{-x}(y^{T_p \to T_q})]$$
$$+ \min_{\forall T_k}[SBox^{xz}(T_j \to T_k) + dynamcost^{-x}(z^{T_l \to T_k})] + \beta \cdot bumpcost(x^{T_i \to T_j}). \quad (5)$$

In the above dynamic programming formulation, subnet x moves to track T_j, and we recursively compute the best track positions of all subnets of n_r starting with the subnets y and z that are adjacent to x. If y makes a transition to track T_q, it incurs a SBox connection cost $SBox^{xy}(T_j \to T_q)$ to connect to x and a transition cost $dynamcost^{-x}(y^{T_p \to T_q})$ for moving from track T_p to T_q.

```
Algorithm B&R(s_i)      /*s_i is the bumped subnet */
Begin
1     TranSet = MinCostCalc(s_i);    /* cost-ordered transition set of subnets (including s_i) of the
      net n_r containing s_i */
2      for j ← 0 to t − 1 /* t is the total # of tracks */
3     ChildList = GetChildList(TranSet[j]);    /* set of subnets occupying the new tracks of sub-
      nets of n_r that are chosen for movement in TranSet[j] */
4          if(ChildList == NULL)              /*there is no bumping */
5               DoUpdates(TranSet[j]);          /* update OG and CG */
6               return(success);       /* Converging transition set found for s_i */
7          endif
8          else begin
9          if any child is an ancestor /* this leads to a cycle */
10          break;     /* take next best transition */
11          DoUpdates(TranSet[j]); /* update OG and CG */
12          numsuccess = 0; /*keep track of number of successful T-DAGs */
13          for each C ∈ ChildList
14               ReturnFlag = B&R(C)
15               if( ReturnFlag == fail) break;
16               else numsuccess++;
17          endelse
18          if (numsuccess == |ChildList| ) /* converging T-DAG for all children found */
19               return(success);
20     endfor
21     return(fail); /* B&R has failed */
End
```

Fig. 11. Bump and refit algorithm to find a converging transition set.

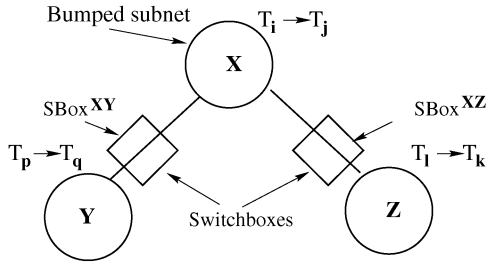

Fig. 12. Subnet X is bumped which is initially on track T_i. Subnets Y and Z are adjacent to X in its CG. $SBox^{XY}$ is the switch box connecting adjacent channels on which subnet X and subnet Y lie.

In computing $dynamcost^{-x}(y^{T_p \rightarrow T_q})$, the subnet x which is the parent of y in this recursive computation tree is not considered for further movement. The *dynamcost* computation for a node terminates when all nodes adjacent to it in its *CG* are its ancestors in the computation tree. $SBox^{xy}(T_j \rightarrow T_q)$ is the minimum cost path within the SBox computed by function MinCostPath to make the $T_j \rightarrow T_q$ connection (see Figure 7), where the cost of each node in the SBox is given by Equation (3). $bumpcost(x^{T_i \rightarrow T_j})$ is given in Equation (4), and the β weighting factor is the same as the one in Equation (3) (which has

```
Algorithm MinCostCalc(s_i)
Begin
for T_j ← 0 to t − 1 ;T_j ≠ T_curr    /*T_curr is current track of s_i, t is
total number of tracks */
      dynamcost( s_i^{T_curr→T_j} ); /* see Equation (5)*/
TranSet[j] = dynamcost ( s_i^{T_curr→T_j}); /*store the track-set for a tran-
sition in an array*/
endfor
sort(TranSet) /* sort the TranSet array in increasing order of cost
*/
return (TranSet);
End
```

Fig. 13. Algorithm to calculate the minimum cost subsection of the bumped net to be perturbed.

```
Algorithm IncrRoute(𝒩)
/* 𝒩 is the set of new/modified nets to be incrementally routed */
Begin
for each n_i ∈ 𝒩 do
    BB = minimal bounding box for n_i;
    repeat
        GRoute(n_i); /* GRoute (Figure 5) implicitly routes within BB */
        (flag, path) = DRoute(n_i); (Figure 7)
        if (flag == fail) then begin
            increase BB by 1 unit on each side;
            if BB cannot be increased further then note n_i as an un-
            routed net;
        endif
    until(flag == success or BB could not be increased);
End
```

Fig. 14. Main algorithm for incremental routing of a set of nets. Note that `DRoute` calls `B&R` (Figure 11) which in turn calls `MinCostCalc` (Figure 13).

another weighting factor α for the switch usage cost that is thus implicitly included within Equation (5)).

Using the recursive function *dynamcost* of Equation (5), Algorithm `MinCostCalc` (Figure 13) computes an array of the t best transition patterns of all subnets of a net n_i, given that a particular subnet s_i of n_i has been bumped. The main algorithm's flow for incrementally routing a set of new/modified nets is given in Figure 14; this pseudocode is a compact version of the flowchart given in Figure 3 and all four algorithms (`GRoute`, `DRoute`, `B&R`, and `MinCostCalc`) discussed in this article are directly or indirectly called in it.

We next establish the optimality of the `MinCostCalc` (Figure 13) algorithm and the near-optimality (optimality under certain conditions) of the `B&R` algorithm (Figure 11).

THEOREM 1. *Algorithm* `MinCostCalc` (*Figure* 13) *for an O-subnet of a net* n_j *returns an optimal subsection solution with respect to the cost metric used in*

Equation (5). *Furthermore, the time complexity of* `MinCostCalc` *is* $O(st^3 \log t)$, *where* s *is the number of subnets of* n_j *and* t *the number of tracks in a channel.*

PROOF. Let x be an *O-subnet* of net n_j and $y \in adj(x)$ in n_j's CG. For each transition T_j of x, where $T_j \neq$ current track of x, *dynamcost* recursively computes the min-cost transition for each y that includes the SBox connection cost to x^{T_j}. Because a net route is a tree (it has no cycles), the transition cost of each y is independent of the transition of its parent x in the dynamic programming computation tree; that is, in Equation (5), $dynamcost^{-x}(y^{T_p \to T_q})$, for example, is independent of the track T_j to which x transits. Thus the transition costs of the ys to different tracks T_q can be computed once and then added to the appropriate $SBox^{xy}(T_j \to T_q)$ to compute the minimum cost subnet to track assignments of y and all its "children" subnets in the computation tree to remain connected to x, when x transits to T_j.

The minimum transition cost of $x^{\to T_j}$ is then the sum of the minimum transition costs of all ys plus the $bumpcost(x^{\to T_j})$ (again, the transitions of each y and its children subnets in the computation tree are independent of the other ys and their children subnets). This is exactly the cost computed in $dynamcost(x^{T_i \to T_j})$ in Equation (5). `MinCostCalc` then orders by increasing cost all the $t-1$ $dynamcost(x^{T_i \to T_j})$ transition costs of the *O-subnet* x; it thus returns an optimal solution as the first cost in the returned cost array.

Because there are $\Theta(t)$ switches and connections in a SBox (and thus in its DGraph), each $SBox^{xy}(T_j \to T_q)$ cost can be computed in $\Theta(t \log t)$ time, using Dijkstra's shortest path algorithm [Cormen et al. 1990] and thus the $\min_{\forall T_q}[SBox^{xy}(T_j \to T_q) + dynamcost^{-x}(y^{T_p \to T_q})]$ computation takes $\Theta(t^2 \log t)$ plus the time to compute $dynamcost^{-x}(y^{T_p \to T_q})$. In as much as there are a constant number of subnets $y \in adj(x)$, the computation of Equation (5) takes $\Theta(t^2 \log t)$ plus the time to compute $dynamcost^{-x}(y^{T_p \to T_q})$ over all $y \in adj(x)$.

Furthermore, over all possible $t-1$ transitions of x it takes `MinCostCal` a time of $\Theta(t^3 \log t)$ plus the time to compute $dynamcost^{-x}(y^{T_p \to T_q})$ over all ys; note that the $dynamcost^{-x}(y^{T_p \to T_q})$ computation needs to be done only once irrespective one of the transitions of x. At a leaf subnet u in the computation tree, it takes $\Theta(t)$ time to compute the *bumpcost* of this subnet to $t-1$ possible tracks, and this is the only computation involved in $dynamcost^{-v}(u^{T_p \to T_q})$ over all T_q, where v is u's parent in the computation tree. If there are k subnets among a y and all its children, then in the worst case (when the subtree rooted at y is a path), it will take $O(t + (k-1)t^3 \log t = O(kt^3 \log t)$ time to compute $dynamcost^{-x}(y^{T_p \to T_q})$ over all T_q. Because the sum of all such ks over all ys, $y \in adj(x)$, is $s-1$, `MinCostCalc` has a complexity of $O(st^3 \log t)$. □

A subsection transition cost (TC) function $f(s_b^{T_j \to T_k}, S(n_i)^{T_j(s_r) \to T_k(s_r)})$, where s_b is the bumped subnet of net n_i, $S(n_i)$ is any subsection of net n_i [set of one or more subnets of n_i] that includes s_b, T_j (T_k) is the current (new) track position of the subnet s_b, and $T_{j(s_r)}$ ($T_{k(s_r)}$) is the current (new) track position of the subnet $s_r \in S(n_i)$, is said to be *trackwise monotonic* if for any two subsections $S(n_i)$ and $R(n_i)$ both of which include s_b, $f(s_b^{T_j \to T_k}, S(n_i)^{T_j(s_r) \to T_k(s_r)}) \leq f(s_b^{T_j \to T_k}, R(n_i)^{T_j(s_r) \to T_k(s_r)})$ implies that if a detailed routing solution exists for

moving each subnet of subsection $R(n_i)$ from its current track as determined by the subsection TC, then a detailed routing solution also exists for moving each subsection of $S(n_i)$ from its current track as determined by the TC. Considering the *dynamcost* subsection transition cost function of Equation (5), it is reasonable to expect that it would be "almost" trackwise monotonic, that is, the above implication holds in most of the cases, but (complete) trackwise monotonicity is, of course, not guaranteed. Our attempt below is to show the optimality of the search process of the B&R algorithm irrespective of the subsection TC it uses to guide the subsection of a bumped net n_i it would actually bump out of its current track(s) and what the new track(s) of this subsection would be. We can establish this optimality if we assume that the TC it uses is trackwise monotonic, and given such a TC, it will find a detailed routing solution if it exists when each time a subnet of net n_i is bumped, B&R moves an optimal subsection $S(n_i)$ (that includes the bumped subnet) from its current track(s).

THEOREM 2. *Let s_b be an O-subnet of net n_i bumped by the routing of a new net n_j. If dynamcost* (*Equation* (5)) *is a trackwise monotonic TC, then Algorithm* B&R (*Figure* 11) *will find a detailed routing solution for bumping s_b out of its current track, if such a solution exists.*

PROOF SKETCH. According to Theorem 1, Min_Cost_Calc returns the minimum cost subsection $\mathcal{S}$ to move for each bumped net n_i' (with bumped subnet s_b') in the B&R search process. If any such move creates a cycle due to one or more subnets of $\mathcal{S}$ bumping into an ancestor in the path formed so far by B&R, then it has been shown (see [Dutt et al. 1999] for the result statement) that if a solution exists, it can be found by a noncyclic traversal (i.e., by a DAG). Hence B&R abandons the current set of transitions of $\mathcal{S}$ (corresponding to, say, s_b' transiting from T_j to T_k) and explores another set of transitions for the optimal subsection returned by Min_Cost_Calc corresponding to a different transition of s_b' from T_j to T_l. We assume that the TC computed by Min_Cost_Calc is trackwise monotonic and we explore the least-cost subsection of n_i for moving (corresponding to s_b' transiting from T_j to T_k), therefore it means that if a detailed routing solution exists for the current path for moving any other subsection of n_i' corresponding to s_b' moving to T_k so that the traversal is acyclic (otherwise the solution will not be found for the current path), then a solution also exists for moving the least-cost subsection $\mathcal{S}$. Currently, this solution path is cyclic as we have assumed; however, this means an acyclic solution exists. Such a solution is found later by B&R by backtracking its current path (if it does not find another solution via other transitions of s_b') as explained below.

If B&R is unable to find a solution for all $t - 1$ track transitions (t is the number of used tracks) it explores for s_b' (and the corresponding least cost subsection transitions), then it backtracks to the net that bumped s_b' and tries a different transition for it. If there exists a solution in which subnet s_b' of n_i' is bumped from T_j to T_k along a different traversal path taken by B&R and in which the corresponding least-cost subsection $\mathcal{S}$ of n_i' is moved, then it will be found by B&R backtracking to or above the highest-level ancestor bumped by the subnet(s) of $\mathcal{S}$ in the current path and choosing a path in which no ancestor is

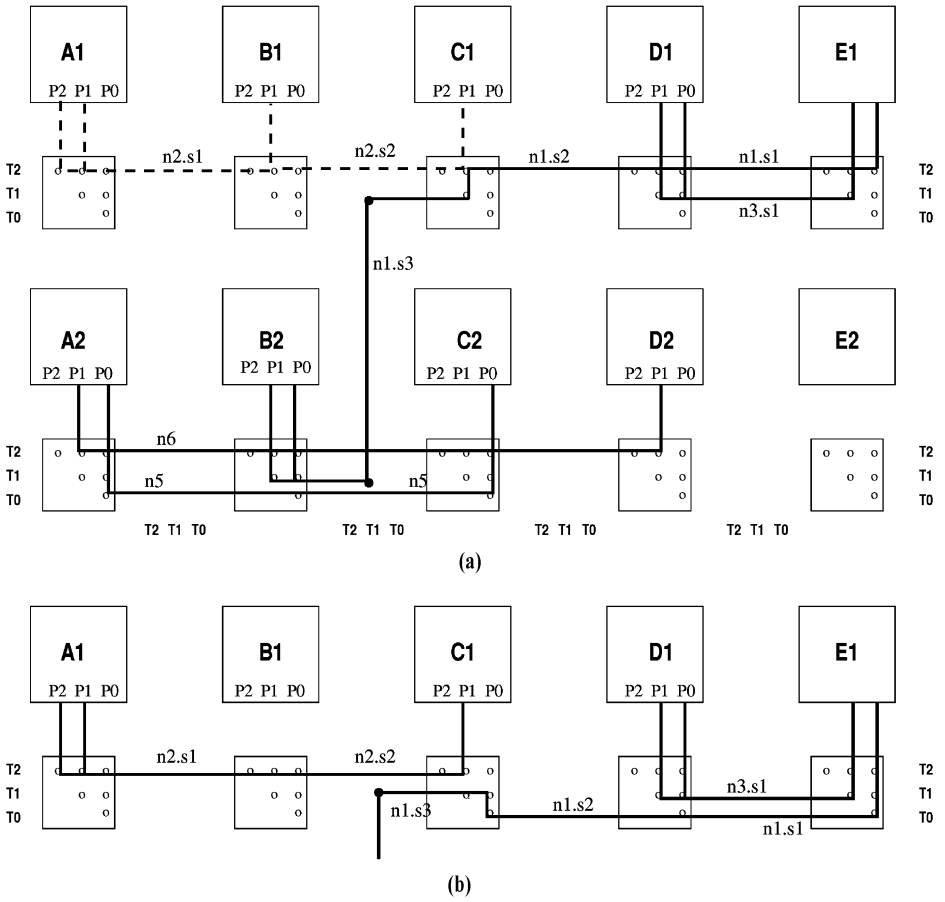

Fig. 15. (a) Net n_2 needs to be rerouted and connected to pin p_1 of PLB C_1; subnet $n_1.s_2$ occupies the track and switch required by net n_2. (b) The final routing.

bumped when it again explores the transition of subsection $\mathcal{S}$. In other words, if there exists a noncyclic traversal solution, B&R will find it by its cycle detection and backtracking mechanism. Because a solution found by a cyclic traversal implies the existence of a noncyclic traversal solution (see Dutt et al. [1999]), and because B&R explores the optimal subsection to move for each transition of bumped subnet s_b, then assuming that the TC used by Min_Cost_Calc is trackwise monotonic, B&R will find a detailed routing solution for routing new net n_j if it exists. □

Note that since in actuality the TC *dynamcost* used by Min_Cost_Calc is probably "almost" trackwise monotonic, but not completely so, we can say that under this condition Algorithm B&R is *near-optimal*.

In Figure 15 $n_1.s_2$ is the O-subnet that needs to move out from track T_2 in order to route n_2 to connect to pin p_1 of PLB C1 via the switch currently occupied by $n_1.s_2$. If $n_1.s_2$ alone moves to track T_1 it will bump $n3.s1$ in order to connect to $n_1.s_1$ ($T_1 \rightarrow T_2$ connection) via the switches used by $n3.s1$; this incurs a high

SBox cost. If $n_1.s_2$ moves to track T_0 it will also similarly incur a high SBox cost to connect track T_0 to T_2. If both $n_1.s_2$ and $n_1.s_1$ are moved to track T_0, then they do not bump into any other net on T_0 or on any switches needed to maintain connection between them; hence a low SBox cost is realized. We also see from Figure 15(a) that if $n_1.s_3$ makes a transition to track T_2 or T_0 it bumps into net n_6 or n_5, respectively, resulting in a high bumping cost. If $n_1.s_3$ remains at its current track, it neither bumps into any net nor incurs a high switching cost to maintain connection to $n_1.s_2$ if the latter moves to T_0; see Figure 15(b). Hence the set $\{n_1.s_2, n_1.s_1\}$ is the optimal subsection to bump, and is returned as the min-cost transition by `MinCostCalc` (Figure 13).

4. EXPERIMENTAL RESULTS

Our B&R incremental routers were tested on Lucent's ORCA-2C FPGA, which has a 10×10 PLB array and on the VPR FPGA architecture [Betz and Rose 1997; Lemieux and Brown 1993] from the University of Toronto.

4.1 ORCA Results

In our current implementations we only use track segments of unit length. The inputs to our software are placed and routed circuits created using Lucent's graphical tools. Nets spanning the range from local to global were created: within the 10×10 PLB array, net lengths ranged from two (local) to 23 (global) spanned PLBs, with the average net length being 7 across all circuits. The experiments were performed on 550 MHz Pentium-III Linux and 167 MHz Sun Ultra 1 Solaris machines. Our initial simulations on `B&R`, `A_PAR`, and `Rip-up&Reroute` were done on the Ultra 1 machine. We then migrated to the Pentium-III machine for later versions of `B&R`; `A_PAR`, however, does not run on Linux, and so based on the `B&R` and `Rip-up&Reroute` run-times on the two machines, we obtained the speed ratio and scaled the `A_PAR` run-time to the Pentium-III equivalent time. In Table I we report the CPU time for the Pentium-III machine (among other parameters). To simulate ECO, we randomly removed 5, 10, and 15% of the original nets, and added the same percentage of new nets with the same number of pins but with random pin positions. The metrics of interest for incremental rerouting for ECO applications are run-time, length and delay of rerouted nets, and change in the electrical properties (such as power and delay) of the circuit. The aim of any incremental router should be to have minimal effect on the rest of the nets and hence minimally affect the electrical properties of the non-ECO portion of the circuit. We compared the two versions of the B&R algorithm to the two prior techniques `Standard` (`Std`) [Emmert and Bhatia 1998] and `Rip-up&Reroute` (`R&R`) [Cong and Sarrafzadeh 2000] implemented by us. All the incremental routing techniques that we experimented with have global and detailed routing phases. In `Std`, if the detailed router is unsuccessful, the bounding box for the global router is increased. In the `R&R`, if the detailed router is unsuccessful, the existing nets occupying the needed routing resources are ripped up and rerouted using global and detailed routing. The nets ripped up are exactly those that the detailed router bumps; we use a similar detailed router as in the B&R approach. This provides very specific

Table I. ECO Routing Simulated for 20 Runs in Each Case[a]

Circuit	PLBs	Nets	A_PAR	%New Nets	Std		R&R			Subnet_B&R		Subsec_B&R	
			time (msec.)		Avg. len	time (msec.)	Avg. len	time (msec)	Ripped-Up %Δlength	Avg. len.	time (msec)	Avg. len	time (msec.)
123.ncl	80	83	1200	5	8.8	74	8.8	70	—	8.8	38	8.8	51
				10	9.5	110	9.18	90	+12.75	9.8	60	9.05	80
				15	9.2	160	8.95	131	+15.75	9.3	110	9.1	120
Eco-Bist2	66	42	550	5	8.1	20	8.1	20	—	8.1	20	8.1	20
				10	9.4	45	8.87	48	—	9.1	37	8.7	39
				15	8.28	59	8.25	67	−3.1	8.22	52	8.15	57
Eco-Big	79	66	1230	5	8.7	50	8.6	48	—	8.6	49	8.6	43
				10	8.6	108	8.2	110	+4	8.5	98	8.4	96
				15	9.4	146	8.71	142	+11	8.8	120	8.69	129
Eco-Bist4	67	47	770	5	7.8	57	7.8	54	—	7.8	51	7.8	51
				10	10.1	90	9.7	83	+7	9.67	30	9.97	27
				15	10.8	101	8.9	98	+30	9.36	42	9.06	46
Eco-Big2	80	84	1380	5	8.6	58	8.4	57	—	8.4	37	8.5	41
				10	9.7	108	9.41	101	+20	9.45	54	8.97	63
				15	10.6	145	8.93	131	+14.5	9.31	88	8.14	108
Average				5	8.4	51.8	8.34	49.8	—	8.34	39	8.36	41.2
				10	9.46	92.2	9.07	86.4	+8.75	9.3	55.8	9.01	61
				15	9.65	122.2	8.75	113	+13.6	9.0	82.4	8.74	92

Avg. len is the average routed length of new nets, and **Ripped-Up %Δlength** is the change in the length of bumped/ripped-up nets in R&R.

information on which nets to rip up which is an advantage for it, and to the best of our knowledge such specific information is not used to target nets for ripping up in other R&R routers.

From Table I we can summarize the following results.

—The Subnet_B&R methodology is the fastest. It is nearly 30% faster than Std and nearly 27% faster than Rip-up&Reroute for the 15% new net case. The Subsec_B&R method is nearly 25% faster than Std and nearly 18% faster than R&R for the above case.

—The Subsec_B&R methodology produces the best solution quality. The average length of the new nets is nearly 12% smaller than Std for the 15% new net case.

—The average length of the new nets is almost the same for the Subsec_B&R and R&R. However, in R&R, the average length of the ripped-up (bumped) nets increased by 13% for the 15% new net case and and by 8% for the 10% new net case. In both B&R techniques, the length of the existing bumped nets does not change, a very significant advantage.

—As expected, the Subnet_B&R is on average 10% faster than Subsec_B&R, but the length of the rerouted nets in Subsec_B&R is nearly 3% smaller than that of Subnet_B&R.

—Our incremental routers are nearly 10 to 20 times faster than Lucent's auto-place and route A_PAR when used in a complete rerouting framework.

4.2 VPR Results

We also implemented our methods as well as the Std and R&R methods for the VPR FPGA from the University of Toronto [Betz and Rose 1997; Lemieux and Brown 1993] for the i-to-i SBox architecture of the type shown in Figure 8. In the B&R implementation for this type of SBox, the entire bumped net has to be moved out of its current track T_i and into a new track T_j; we call this implementation Fullnet_B&R[3] (B&R for short whenever there is no confusion about which B&R version we are referring to), which, similar to the case of a CS-insertion in Dutt et al. [1999], is optimal in terms of finding a detailed routing for a given global route of a net, if one exists (a stronger result than that of Theorem 2). Two types of cases were simulated for the VPR FPGA: availability of 10% unused tracks in the FPGA after the initial routing of the circuit; and the really hard case of 0% unused tracks. The initial routings of the circuits were performed using the VPR router and then the three incremental routers Std, R&R, and B&R were used to reroute 5 and 10% of the nets in 20 random runs as in the previous simulations for Lucent's ORCA FPGA. Because the Std and R&R methods especially could not route an appreciable number of nets (the unroutable numbers for B&R are miniscule in the 10% unused tracks case and significantly less than those of the other methods in all cases), we include in our results the number of unrouted nets and pins for each circuit by the three different methods. Also, for the same reason, it does not make much

[3]In Fullnet_B&R, we actually use a very fast version of the basic B&R algorithm of Figure 11, which prunes the B&R search space without sacrificing optimality; see Arslan and Dutt, 2002.

Table II. Characteristics of VPR Circuits Used in Our Experiments

Circuit	PLBs	Nets	Pins	Pin density	Circuit	PLBs	Nets	Pins	Pin density
sse	121	82	291	3.77	s838.1	144	149	404	3.45
rd73	144	103	379	3.87	ex1	196	149	557	3.81
pma	144	108	398	3.94	s820	169	156	542	3.79
cse	144	110	410	3.87	mult32a	169	161	483	3.45
sao2	144	114	412	3.85	clip	196	171	639	3.87
mm4a	144	118	390	3.71	i5	729	231	378	3.09
term1	144	134	395	3.66	example2	441	237	596	3.44
s713	144	147	395	3.4	i4	729	292	384	2.72

sense to compare the old and new lengths of ripped-up nets in the R&R method (many of the longer ripped-up nets cannot be rerouted in R&R) and thus those data are not included in our results here. Instead, we obtain the percentage difference %ΔhpBB in the half-perimeters (HPs) of the pin BB and the routing BB (the former is a lower bound for the latter) for each new and ripped-up (the latter applies only to the R&R method) net as a measure of how effective the different methods are in incrementally finding minimal-length routes. However, this metric is really overridden by the unrouted nets/pins metric as, once again, if a particular method is unable to find routes for a number of, generally, longer nets, it is not very meaningful that it is able to route the shorter nets effectively. In spite of this, we still include the %ΔhpBB metric for completeness.

The results are tabulated in Tables III to VI; the circuit characteristics are given in Table II. In each table the VPR time to route the original circuit is included as an approximate measure of the time it would take to reroute the changed circuit completely. Unfortunately, we could not get VPR to actually reroute the changed circuit, since the changes we introduced were random, which left some PLBs whose inputs or outputs were not connected and these resulted in error messages from VPR; it would have been interesting to measure the routing completion effectiveness of VPR.

The salient points of the experimental results are as follows.

—The B&R method is at least an order of magnitude more effective in routing completeness for the 10% unused tracks case: it has 50 (≈20) and 330 (≈85) times fewer unrouted nets than Std and R&R, respectively, for the 5% (10%) new nets case. For the hardest case of 0% unused tracks, the gap between B&R and the other two methods reduces mainly due to the fact that even optimally it is not possible to route an appreciable fraction of new nets (12.5 to 15% unrouted nets for B&R which is optimal in the detailed routing phase). The factor differences between B&R, and Std and R&R are about 1.7 (1.5) and 10 (3.5), respectively, for the 5% (10%) new nets case. Surprisingly, R&R performs worse than even Std on this metric.
On an absolute scale, B&R is able to complete the required new net routes almost every time for the 10% unused tracks case (successful in 15 out of 16 circuits for the 5% new net case, and for 12 out of 16 circuits for the 10% new net case; furthermore, it is unsuccessful for the other circuits for a miniscule number of nets out of the 20 random runs that were performed for deleting old nets and adding in the new ones), whereas, for almost all circuits, the

Table III. Results for Three Incremental Routing Methods for the VPR FPGA Architecture[a]

Ckt.	VPR	New Nets	Std				R&R				Fullnet_B&R			
	time (sec.)		unrt. nets	unrt. pins	%ΔhpBB	time (sec.)	unrt. nets	unrt. pins	%ΔhpBB	time (sec.)	unrt. nets	unrt. pins	%ΔhpBB	time (sec.)
sse	2.97	4	0.2	1.25	0.038	0.001	0	0	0.037	0.000	0	0	0.038	0.03
rd73	4.12	5	0.2	0.55	0.04	0.003	1.05	17.35	0.048	0.039	0	0	0.04	0.06
pma	3.79	5	0	0	0.02	0.001	1.2	21.9	0.013	0.047	0	0	0.02	0.09
cse	3.67	5	0.1	1	0.06	0.001	1.15	20.15	0.052	0.024	0	0	0.06	0.04
sao2	4.38	5	0.05	0.1	0.04	0.002	0.75	10.15	0.043	0.023	0	0	0.04	0.06
mm4a	3.81	5	0.15	0.45	0.01	0.003	1.45	10.05	0.01	0.049	0	0	0.01	0.06
term1	3.55	6	0.1	0.4	0.07	0.003	0.3	2.75	0.058	0.009	0	0	0.067	0.05
s713	3.54	7	0.1	0.3	0.057	0.001	0.5	2.4	0.06	0.023	0	0	0.057	0.06
s838.1	2.31	7	0.05	0.2	0.014	0.002	0	0	0.014	0	0	0	0.014	0.05
ex1	5.69	7	0.4	2.4	0.036	0.004	3.05	49.5	0.05	0.082	0.05	0.03	0.036	1.41
s820	5.63	7	0.15	0.4	0.057	0.005	1.1	20	0.071	0.03	0	0	0.057	0.08
mlt32	2.13	8	0.2	0.6	0.046	0.001	0.25	1	0.041	0.012	0	0	0.044	0.06
clip	7.22	8	0.35	1.95	0.038	0.005	2.1	42.9	0.035	0.06	0	0	0.038	0.71
i5	6.28	11	0.2	0.6	0.075	0.009	0	0	0.077	0.008	0	0	0.073	0.16
exmp2	14.94	11	0.25	0.65	0.088	0.008	2.85	34.75	0.08	0.081	0	0	0.082	0.18
i4	6	14	0.1	0.2	0.054	0.02	0	0	0.052	0.011	0	0	0.054	0.21
Avg.	5.002	7.19	0.16	0.69	0.05	0.004	0.98	14.55	0.05	0.03	0.003	0.019	0.045	0.21

[a]With 5% new nets and 10% unused tracks for 20 random runs per circuit. The salient part of these results is the number of unrouted nets and pins for the different methods.

Table IV. Results for Three Incremental Routing Methods for the VPR FPGA Architecture[a]

Ckt.	VPR	New Nets	Std				R&R				Fullnet_B&R			
	time (sec.)		unrt. nets	unrt. pins	%ΔhpBB	time (sec.)	unrt. nets	unrt. pins	%ΔhpBB	time (sec.)	unrt. nets	unrt. pins	%ΔhpBB	time (sec.)
sse	2.97	8	0.35	1.3	0.02	0.004	1.65	20.6	0.02	0.043	0	0	0.019	0.47
rd73	4.12	10	0.65	2	0.037	0.007	2.05	45.45	0.027	0.06	0	0	0.036	3.36
pma	3.79	10	0.55	3	0.02	0.004	1.45	24.55	0.025	0.05	0	0	0.020	0.17
cse	3.67	11	0.5	1.85	0.034	0.004	3.05	51.7	0.022	0.077	0	0	0.032	1.34
sao2	4.38	11	0.4	1.4	0.037	0.005	2.4	36.35	0.038	0.07	0.05	0.25	0.036	0.21
mm4a	3.81	11	0.6	1.9	0.069	0.006	3.4	23.9	0.045	0.124	0	0	0.064	6.2
term1	3.55	13	0.55	1.8	0.054	0.004	0.55	6.35	0.046	0.021	0	0	0.051	0.09
s713	3.54	14	0.4	1.2	0.081	0.003	1.55	12.9	0.076	0.055	0	0	0.079	1.27
s838.1	2.31	14	0.2	0.8	0.036	0.005	0.2	1.15	0.031	0.011	0	0	0.036	0.09
ex1	5.69	14	0.8	3.65	0.05	0.006	4.15	72.85	0.044	0.098	0.25	1.7	0.047	1.81
s820	5.63	15	0.6	1.65	0.059	0.007	3.1	59.65	0.063	0.089	0	0	0.057	1.42
mlt32	2.13	16	0.4	1.35	0.054	0.006	0.25	10.6	0.052	0.01	0.05	0.15	0.05	0.1
clip	7.22	17	1.3	6.4	0.034	0.006	6.7	119.2	0.029	0.231	0	0	0.032	18.02
i5	6.28	23	0.35	0.95	0.057	0.021	0.45	1.6	0.067	0.041	0	0	0.057	0.29
exmp2	14.94	23	0.75	2.6	0.058	0.018	3.55	41.95	0.061	0.121	0.05	0.15	0.057	3.11
i4	6	29	0.4	0.8	0.047	0.035	0.05	0.1	0.045	0.028	0	0	0.047	0.39
Avg.	5.002	14.94	0.55	2.04	0.047	0.009	2.16	33.056	0.043	0.07	0.025	0.14	0.045	2.4

[a]With 10% new nets and 10% unused tracks for 20 random runs per circuit. The salient part of these results is the number of unrouted nets and pins for the different methods.

Table V. Results for Three Incremental Routing Methods for the VPR FPGA Architecture[a]

Ckt.	VPR	New Nets	Std				R&R				Fullnet_B&R			
	time (sec.)		unrt. nets	unrt. pins	%ΔhpBB	time (sec.)	unrt. nets	unrt. pins	%ΔhpBB	time (sec.)	unrt. nets	unrt. pins	%ΔhpBB	time (sec.)
sse	2.97	4	1	4.1	0.042	0.005	5.85	64.15	0.042	0.13	0.8	3.15	0.042	3.22
rd73	4.12	5	1.45	4.6	0.063	0.007	7.2	109.15	0.016	0.24	1.3	4.15	0.063	7.97
pma	3.79	5	1.1	3.15	0.03	0.006	7.35	111.05	0.018	0.25	0.9	2.5	0.00	11.78
cse	3.67	5	1.25	4.7	0.098	0.009	6	91.95	0.022	0.14	1	3.45	0.097	4.47
sao2	4.38	5	1.2	3.4	0.045	0.007	7.6	100.6	0.026	0.22	0.05	0.25	0.037	32.89
mm4a	3.81	5	1.6	4.6	0.013	0.008	13.05	91.35	0.019	0.37	1.35	3.9	0.01	26.11
term1	3.55	6	1.2	3.15	0.089	0.006	5.3	45.5	0.044	0.13	0.5	1.35	0.077	21.04
s713	3.54	7	1.7	4.1	0.071	0.008	9.35	71.8	0.05	0.27	1.2	2.85	0.063	22.92
s838.1	2.31	7	0.95	2.95	0.017	0.008	2.35	10.4	0.01	0.07	0	0	0.014	0.21
ex1	5.69	7	2.7	10.8	0.035	0.012	12.1	188.25	0.049	0.29	0.1	0.2	0.02	12.54
s820	5.63	7	2.25	7.5	0.059	0.013	7.4	141.85	0.057	0.18	1.7	5.9	0.059	8.46
mlt32	2.13	8	1.55	5.15	0.04	0.007	2.95	48.9	0.035	0.07	0.6	2.35	0.039	0.29
clip	7.22	8	2.55	9.4	0.062	0.014	11.4	198.8	0.02	0.34	2	7.8	0.058	70.35
i5	6.28	11	1	2.6	0.085	0.029	3.25	11.3	0.104	0.13	0.25	0.65	0.083	0.19
exmp2	14.94	11	2.55	6.95	0.139	0.045	11.25	125.25	0.068	0.31	1.65	4.65	0.129	19.17
i4	6	14	0.9	1.8	0.057	0.035	1.15	2.3	0.067	0.79	0.4	0.8	0.062	0.18
Avg.	5.002	7.19	1.56	4.93	0.06	0.013	7.1	88.29	0.04	0.2	0.89	2.88	0.053	15.11

[a]With 5% new nets and 0% unused tracks for 20 random runs per circuit. The salient part of these results is the number of unrouted nets and pins for the different methods.

Table VI. Results for Three Incremental Routing Methods for the VPR FPGA Architecture[a]

Ckt.	VPR	New Nets	Std				R&R				Fullnet_B&R			
	time (sec.)		unrt. nets	unrt. pins	%ΔhpBB	time (sec.)	unrt. nets	unrt. pins	%ΔhpBB	time (sec.)	unrt. nets	unrt. pins	%ΔhpBB	time (sec.)
sse	2.97	8	2.85	9.2	0.02	0.014	7.45	82.1	0.033	0.2	2.3	7.3	0.016	7.74
rd73	4.12	10	3.7	11.45	0.04	0.02	9.3	145.7	0.022	0.3	3.1	10.2	0.039	16.51
pma	3.79	10	2.95	11.9	0.013	0.013	7.45	114.85	0.018	0.25	1.95	8.2	0.012	25.51
cse	3.67	11	3.75	11.8	0.062	0.016	8.1	119.35	0.016	0.19	2.7	8.35	0.042	7.75
sao2	4.38	11	3.05	9.3	0.036	0.016	7.65	108.15	0.03	0.21	1.95	6.95	0.024	22.94
mm4a	3.81	11	3.75	10.7	0.08	0.022	14.1	96	0.026	0.42	2.8	8.75	0.065	32.04
term1	3.55	13	3.35	10.45	0.05	0.018	8.2	73.35	0.042	0.22	1.65	5.45	0.044	50.63
s713	3.54	14	3.55	9.45	0.11	0.017	10.75	85.05	0.053	0.29	2.5	6.7	0.106	20.53
s838.1	2.31	14	2.7	7.65	0.039	0.016	4.85	23.25	0.015	0.16	0.1	0.35	0.032	1.2
ex1	5.69	14	5.75	21.1	0.058	0.019	9.4	163.35	0.06	0.26	5.5	23.05	0.044	21.75
s820	5.63	15	4.6	14.05	0.085	0.021	12.6	193.05	0.055	0.33	3.25	10.05	0.075	16.24
mlt32	2.13	16	3.15	9.75	0.049	0.017	5.05	72.3	0.04	0.14	1.7	5.05	0.04	0.72
clip	7.22	17	6.8	26.2	0.038	0.039	12.3	210.45	0.026	0.35	3.85	21.15	0.022	98.75
i5	6.28	23	2.55	6.3	0.063	0.055	5.6	20.3	0.09	0.204	0.7	1.75	0.054	0.36
exmp2	14.94	23	5.4	15.95	0.071	0.063	12.35	134.7	0.064	0.37	3.45	10	0.06	38.3
i4	6	29	2.95	5.9	0.059	0.094	2.25	4.5	0.065	0.19	1.65	3.3	0.059	0.35
Avg.	5.002	14.94	3.80	11.95	0.054	0.028	8.59	102.9	0.04	0.26	2.44	8.54	0.046	22.58

[a]With 10% new nets and 0% unused tracks for 20 random runs per circuit. The salient part of these results is the number of unrouted nets and pins for the different methods.

other two methods are not able to complete all the required reroutes for all 20 random runs. This underscores the significantly greater efficacy of the B&R approach in routing nets incrementally over those of the other methods.

—In the %ΔhpBB metric, B&R is consistently better than Std in all cases. R&R *seems to be* slightly better than B&R for the 0% unused tracks cases, but this is really an erroneous conclusion as R&R has a significantly larger number of unrouted nets than B&R. For the 0% unused tracks case, the factor difference in unrouted nets between Std and B&R is much smaller (with B&R having fewer unrouted nets) and thus the %ΔhpBB comparison between these two methods is more apples-to-apples. The comparison reveals that B&R has about a 15% smaller %ΔhpBB metric than Std in this case; this highlights the greater efficacy of B&R in minimal-length routing than the other two incremental routing techniques.

—B&R is, however, significantly slower than both R&R and Std, although in absolute terms it is quite fast for the 10% unused track case. Furthermore, for the 5% (10%) new net, 10% unused track case, B&R is about 25 times (two times) faster than VPR (used in full re-routing mode).
However, for the more difficult cases, B&R becomes slower than even VPR. These timing results for B&R are in contrast to the timing results for the ORCA FPGA, where it is the fastest method and is orders of magnitude faster than complete rerouting using Lucent's A_PAR router. The reason for B&R's timing turnaround for the VPR architecture is that we used i-to-i SBox's wherein each net has to be routed on the same track in all its channels. This, of course, means that B&R bumps many more nets and thus has large numbers of converging T-DAGs to search for compared to the searches that would be required in FPGAs with i-to-j SBox's such as the ORCA, where there are many more opportunities to avoid bumping existing nets by routing the current net on different tracks. Because almost all current commercial FPGAs have i-to-j switches, B&R would be much faster for these FPGAs as has been demonstrated in Table I for the ORCA FPGA.[4] Finally, it also needs to be noted that since routing incompleteness is significantly higher for Std and R&R compared to B&R, the fact that the former two methods are much faster than B&R is not very meaningful.

5. CONCLUSIONS AND FUTURE WORK

The B&R approach of Dutt et al. [1999] proposed in the context of simple net extensions for fault tolerance was significantly extended, and new concepts such as bumping costs during global routing and optimal bumping subsets of nets were developed in this article to realize a novel and effective incremental routing methodology for ECO applications in FPGAs. Our incremental routers for the ORCA FPGA, which has i-to-j SBox's, are nearly 27% faster and yield nearly 10% shorter net lengths than other proposed incremental routing methods. For our implementations for the VPR FPGA with i-to-i SBox's, the salient result

[4]The i-to-i routing restriction also affects R&R in terms of the number and length of nets that are ripped up and probably leads to its high number of routing incompletions.

is the significantly smaller routing failure rate for the B&R method compared to previous incremental routing techniques, which underscores B&R's near-optimality in completing net routes within the given resources. Although the B&R approach is much slower than the other methods (but reasonably fast in an absolute sense for two of four cases we simulated) for this FPGA architecture, this is primarily due to the inherently larger search spaces imposed on B&R by the routing restrictions in i-to-i SBox's. B&R should, however, be quite fast for current commercial FPGAs that generally have i-to-j SBox's, as demonstrated for the ORCA. Furthermore, our incremental routers do not change the lengths or topologies of non-ECO nets, and hence have minimal effect on the electrical properties of the non-ECO parts of the circuit. Our new B&R routers thus offer significant advantages in almost all metrics of interest to incremental routing in FPGAs. Our experiments also establish the importance of incremental rerouting over complete rerouting. In future work we will incorporate timing-driven aspects in the B&R process. We will also extend this work to regular VLSI chips.

ACKNOWLEDGMENTS

We thank the anonymous referees whose comments helped improve the article.

REFERENCES

ARSLAN, H. AND DUTT, S. 2002. An order-impervious optimal detailed router for FPGAs: depth-first search and optimality-preserving speedup methods. Technical Report, University of Illinois at Chicago, Dec. Available at www.ece.ujc.edu/~dutt/papers/bandrspdup.pdf.

BETZ, V. AND ROSE, J. 1997. VPR: A new packing, placement and routing tool for FPGA research. In *Proceedings of the Seventh International Workshop on Field-Programmable Logic and Applications* (London), 213–222.

CONG, J. AND SARRAFZADEH, M. 2000. Incremental physical design. In *Proceedings of the International Symposium on Physical Design* (April), 84–92.

CONG, J., FANG, J., AND KHOO, K. 1999. An implicit connection graph maze routing algorithm for ECO routing. In *Proceedings of the IEEE International Conference Computer-Aided Design* (April), 12–18.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST R. L. 1990. Introduction to Algorithms. McGraw-Hill, New York.

DUTT, S., SHANMUGAVEL, V., AND TRIMBERGER, S. 1999. Efficient incremental rerouting for fault reconfiguration in field programmable gate arrays. In *Proceedings of the IEEE International Conference Computer-Aided Design*, 173–176.

EMMERT, J. M. AND BHATIA, D. 1998. Incremental routing in FPGAs. In *Proceedings of the IEEE International ASIC Conference and Exhibit*.

HANCHEK, F. AND DUTT, S. 1998. Design methodologies for tolerating logic and interconnect faults in FPGAs. *IEEE Trans. Comput.* Special Issue on Dependable Computing (Jan.).

LEMIEUX, G. AND BROWN, S. 1993. Detailed router for allocating wire segments in FPGAs. In *Proceedings of the ACM Physical Design Workshop* (April), 215–226.

Received April 2002; accepted September 2002