ROAD : An Order-Impervious Optimal Detailed Router for FPGAs*

Hasan Arslan and Shantanu Dutt
Dept. of ECE, University of Illinois-Chicago
Chicago, IL 60607
{harslan, dutt}@ece.uic.edu

Abstract : **It is well known that the solution quality of the detailed routing phase is heavily influenced by the order in which nets are routed. To alleviate this situation a number of routing strategies have been developed that ripup and reroute (R&R) previously-routed nets that "block" the current net. In the R&R approach, there is not a significant amount of control over the solution quality (e.g., length, delay) for the ripped-up nets. In this paper we propose a detailed router *ROAD (bump&*R*efit based* O*ptim*A*l* D*etailed router)* that explores the solution space using an approach called bump-and-refit (B&R) in which the global routes of prior-routed nets are not changed but their track assignments are systematically altered in order to make space for the current net being routed. B&R thus does not have the above drawback of R&R. We start with an initial depth-first search method for this purpose that is optimal in finding a detailed routing solution with the minimum number of tracks irrespective of the net routing order. We then develop various optimality-preserving speedup methods including search space pruning based on clique detection and learning about and remembering unsuccessful search spaces, and second-level or lookahead transition costs. The combination of these methods results in an average speedup of 604 for small to medium VPR circuits and an extrapolated speedup of more than 5763 for larger circuits. Furthermore, comparison of ROAD run times to that of VPR's estimated detailed routing phase show that we are almost two times faster than VPR. This is noteworthy because an optimal detailed router is able to obtain solutions in reasonable times which are also faster than those of a non-optimal (though effective) router.**

1 Introduction

Efficient routing is important for the purpose of reducing total wiring area and/or the lengths of critical-path nets for performance optimization. Both metrics are impacted by the detailed routing phase. A major impediment to effective detailed routing for FPGAs is the net ordering problem, wherein the realization of the global routes of nets within the routing track resources is heavily dependent on the order in which the nets are detailed-routed [9, 7, 10]. A similar problem exists for maze routing in VLSI chips [11, 8, 12]. Though a number of novel approaches using the general approach of *ripup and reroute (R&R)* of previously routed nets that "block" or "collide" with the current net has been proposed and developed [4, 3, 5, 6], the problem has not yet been completely solved for either routing environments.

In this paper, we introduce a new approach to detailed routing that uses a *bump-and-refit (B&R)* strategy to solve the detailed routing problem for FPGAs with *i*-to-*i* switchboxes optimally (i.e., detail route a set of given global net routes of a circuit in the minimum number of tracks) irrespective of the order in which the nets are routed. The B&R approach along with a depth-first search (DFS) algorithm was proposed originally in [1, 2] for the purpose of incremental routing. A similar B&R approach can be used for complete detailed routing, since the routing of the current net in the presence of the previously routed nets is qualitatively an incremental routing problem. Quantitatively, however, truly incremental routing versus the application of an incremental routing model to complete routing differ significantly in the frequency of application of the basic incremental router. In the former, it may need to be applied to 1-10% of the nets, while in the latter, it is applied to almost 100% of the nets. A straightforward application of the DFS B&R incremental algorithm of [1, 2] to the complete routing problem results in slow to extremely slow solution times on a set of medium to large VPR circuits. To alleviate this problem, we develop here a suite of optimality-preserving speedup methods that results in speedups of several orders of magnitude. We thus obtain a detailed router that is optimal as well as time-efficient. It is also noticeably faster than VPR's detail routing phase [13], which implies that the run-time of our router is quite reasonable even though it is optimal.

The rest of the paper is organized as follows. The motivation behind the B&R approach to complete routing is given in Sec. 2. Section 3 discusses the basics of the B&R methodology proposed in [1, 2] and explains how such a methodology can be used to realize an order-impervious optimal detailed router for FPGAs (in the rest of the paper we assume FPGAs with *i*-to-*i* switchboxes as also assumed by the VPR router). Optimality-preserving speedup methods for the B&R DFS algorithm are then developed in Sec. 4. Experimental results for a set of small to large VPR benchmark circuits are given in Sec. 5 and we conclude in Sec. 6.

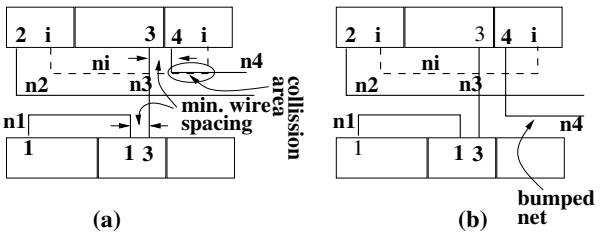


Figure 1: *Advantages of B&R-based routing; new net n_i is shown in dashed lines: (a-b) routing completion for new net n_i in 3 tracks using B&R.*

2 Motivation for B&R

Figure 1 shows completed routings of nets n_1 to n_4 in a channel routing scenario, and the requirement to route another net n_i that cannot be routed even with doglegs unless some of the previous net routings are undone or equivalently if the nets had been routed in a different order, say, $(n_1, n_2, n_3, n_i, n_4)$; Fig. 1b shows a successful routing of all nets in this order. Thus to be able to route nets in an optimal and order impervious manner, a routing algorithm should be able to reverse routing decisions made for previously routed nets. Note that reversal of routing decisions on an existing net n_j in the detailed routing context is equivalent to "bumping" n_j from its current track to another track to make room

*This work was supported in part by NSF grant CCR-0204097.

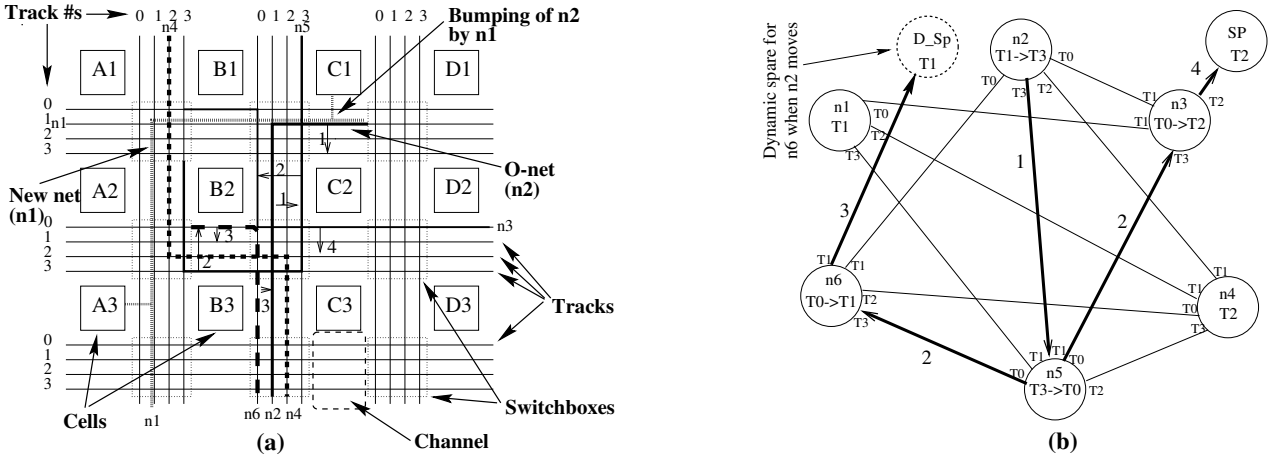


Figure 2: *(a) Routing in an FPGA, and a B&R process for new net n_1 connecting cells A3 and C1; for simplicity pin connections of existing nets are not shown. Nets are shown by dark or dotted lines on the tracks. Numbered arrows from existing nets show the sequence of bumpings to accommodate the routing for n_1. (b) Searching the OG for a converging transition DAG for the O-net n_2. Track labels at each end of an edge in the OG refers to the track on which the neighboring net corresponding to that edge lies.*

for the new net(s) so that a more efficient fitting of nets to tracks is obtained. Thus, for example, the routing solution of Fig. 1b can be obtained from the configuration of Fig. 1a by new net n_i bumping net n_4 which is then moved to the bottom track. Thus the ability to bump previously routed nets can circumvent the net ordering problem. Furthermore, B&R does not change the topology of existing nets while finding an optimal detailed routing solution for a new net (see Sec. 3), and thus preserves their electrical properties to a large degree[1].

3 Bump-and-Refit Algorithms for FPGAs

We first define some terminology pertaining to FPGAs; Figure 2a illustrates many of these terms. We define a *channel* in an FPGA as the set of all track segments between two adjacent *switchboxes (SBoxes)* of the FPGA; see Figure 2a. Each channel has the same number t of tracks, which we denote by $T_0, T_1, \ldots, T_{t-1}$. The length of a *track segment* is the number of channels it spans before it needs to connect to another segment via a SBox. For simplicity of exposition, we describe our B&R algorithm for FPGAs with track segments of length one and SBoxes with i-to-i inter connection capabilities.

3.1 Basic B&R

For setting the stage for applying B&R to complete routing, we discuss here the basic B&R approach and relevant concepts for incremental routing. The "incremental routing problem" can be stated as follows. *Given a circuit placement and routing of existing nets on an FPGA, and a set $\mathcal{N}$ of new nets to be routed, accomplish the routings of the new nets in the minimum number of tracks without changing the global topology of the existing nets.*

For each new net n_i to be incrementally routed, if the required track segments decided by the detailed router is vacant, then no existing nets are disturbed. Note that the detailed router will always prefer to use vacant track segments in the channels assigned to n_i. However, if this is not possible, then it may use one or more segments that are occupied by other net(s), and the routing of n_i will cause a "bumping" of these set of nets called the *occupying set*; each net in this set is called an *occupying net (O-net)*. In Figure 2a, n_1 is a new net connecting cells A3 and C1. Note that it has been routed in a way to minimize bumpings and that there is no shortest route possible for n_1 that uses only vacant track segments. The given routing of n_1 bumps only one net n_2; it is thus the only O-net. In general, refitting solutions for each O-net can be explored independently and in any order to arrive at a feasible refitting of all bumped nets (see Theorem 1 given later). In the sequel, without loss of generality, we thus describe the B&R process for a single O-net.

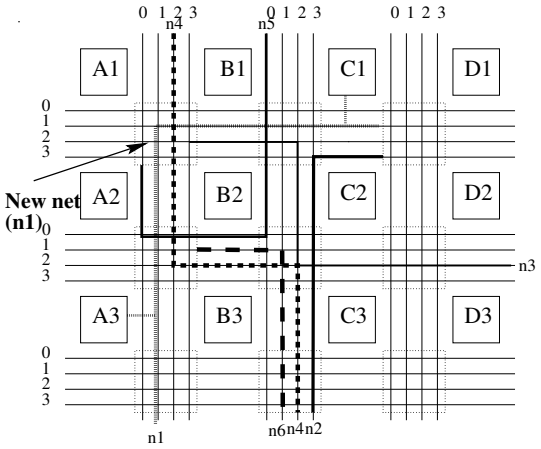


Figure 3: *Final feasible routing based on the converging T-DAG of Figure 2b.*

The O-net needs to be moved out of its current track to make space for the new net. We use $n_i^{T_j}$ to denote a net n_i on track T_j. Let a *transition* be defined as the movement of net n_i on a track T_j to another track T_k, and is denoted by $n_i^{T_j \to T_k}$. This transition may result in net n_i bumping into one or more nets on track T_k. These nets in turn will have to move out of their current track T_k, giving rise to a transition for each of them. This transition sequence is shown in Figure 2b by dark arcs, where net n_2 initiates a set of transitions which finally terminate in "spare" track segments, which are vacant segments of appropriate total lengths into which bumped nets can move without bumping other nets.

[1] In the context of FPGA routing, if the FPGA has segmented tracks, then when a net is bumped to a different track which may have a different segmented structure, it will traverse a different number of switchboxes across its length. Thus during the B&R process, even though a bumped net retains its global topology, a few properties like delay and power can change by a small factor. For FPGAs with non-segmented tracks, however, all electrical properties are preserved.

```
Algorithm Conv-T-DAG(OG, n_i^{T_j})
/* Find a converging T-DAG rooted at n_i^{T_j} */
for T_k=(T_{i_1},....,T_{i_t}) in order of increasing TC do begin
    if (adj^{T_k}(n_i) == ∅) then  /*there exists a spare for n_i on T_k*/
        return (success)
    else begin
        for each n_r ∈ adj^{T_k}(n_i) do begin
            if (n_r is an ancestor) then break;
            /*this transition to T_k results in a cycle, or
            there is a failure try the next best transition */
        for each n_r ∈ adj^{T_k}(n_i) do
            result=Conv-T-DAG(OG, n_r^{T_j});
            if (result==fail) then break;
            else numb_succ=numb_succ+1;
        endfor
            if (numb_succ==|adj^{T_k}(n_i)|) then
                    return(success);
/*converging T-DAGs were found for all nets in adj^{T_k}(n_i) */
    endelse
endfor
return(fail); /* no transition of n_i was successful */
End. /* Conv-T-DAG() */
```

Figure 4: *An optimal depth-first search algorithm for finding a converging T-DAG*

As seen in the Fig. 2b, the set of transitions take on a directed-acyclic graph (DAG) structure, termed a *transition DAG (T-DAG)*, with the spares forming the leaf nodes. The new net routing is successful if a T-DAG rooted at the corresponding O-net can be found whose leaves are spare segments; such a T-DAG is termed a *converging T-DAG*.

We introduce the concept of an *overlap graph (OG)*, which is a graph representation of the circuit routing in the FPGA. The OG is an undirected graph with the circuit nets represented by the nodes of the graph. In the overlap graph OG(V,E), the set of nodes $V = S \cup \{n_1, n_2, \ldots, n_m\}$, where each n_i is a routed net of the circuit and S is the set of "spare" track segments as described above (please note that a set of vacant segments of a track T_j is a spare only with respect to specific net(s) that can be moved to T_j without bumping any nets on that track). There exists an edge between n_i and n_j in the OG if nets n_i and n_j share a channel in the FPGA. Figure 2b shows the OG for the routing of Fig. 2a. In the OG, for e.g., nets n_2 and n_6 have an edge between them since they are routed through a common vertical channel to the right of cell B3.

The OG can be used to determine if the required converging T-DAG exists. Since 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 Fig. 2b). Thus a converging T-DAG rooted at an O-net can be determined by performing a search on the OG until all leaf nodes of the search DAG are spare nodes. 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 by D_Sp in Figure 2b) for net n_6. 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 2b and numbered chronologically in the order in which they are traversed in the search process. Figure 3 shows the final routing of the FPGA after the bumping sequence converges.

3.2 An Optimal Depth-First B&R Algorithm

A B&R incremental routing algorithm Conv-T-DAG (Figure 4) that performs a depth-first based search in the OG for a converging T-DAG rooted at the O-net was developed in [2]. For a transition of the O-net to some track T_k, it recursively searches for converging T-DAGs rooted at each net on T_k bumped by the O-net. A depth-first path terminates in success if a spare node is reached, and in failure either when all OG nodes have been visited in that path or a cycle is detected (an ancestor along the current path is revisited). When an $n_i^{T_j \to T_k}$ transition fails in this manner, the search backtracks and tries an unexplored transition $n_i^{T_j \to T_l}$. The following result (stated slightly differently from that in [2]) establishes the optimality of Conv-T-DAG [2].

Theorem 1 *[2] If converging T-DAGs rooted at the O-nets bumped by a new net routing exist among the currently used tracks of the FPGA, they will be found by calling* Conv-T-DAG *for each O-net in any order.*

While an existing converging T-DAG will ultimately be found by Conv-T-DAG, 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) measures evaluated are as follows:

(1) $TC_1^{sum}(n_i^{T_j \to T_k}) = \sum_{n_j \in adj^{T_k}(n_i)} l(n_j)$,

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 heuristic 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 3 nets each of length 3. However, the latter case has a higher likelihood of convergence since there is greater flexibility in moving 3 bumped nets than a single net of the same total length. This leads to the next cost function.

(2) $TC_1^{sqrt}(n_i^{T_j \to T_k}) = [\sum_{n_j \in adj^{T_k}(n_i)} l(n_j)] / \sqrt{|adj^{T_k}(n_i)|}$.

Using such TC functions to guide the search results in time-to-solution reduction by an order of magnitude compared to a "blind" depth-first search [2]. We term the above TC functions as *1st-level TC functions*.

Time Complexity The worst-case complexity of Conv-T-DAG is equal to the maximum number of paths in the OG starting from any node u, since in the worst-case all paths starting from the O-net will be explored in the DFS process in the algorithm. Let b be the branching factor (average number of neighbors of a node) of the OG, m the number of nodes in it (i.e., number of nets in the circuit), and L the length of the longest path in the OG (e.g., if the OG is a balanced tree then $L = \Theta(\log m)$, and if the OG is a 2-dimensional mesh or a complete graph, then $L = m$). In order to determine the number of paths from u, we can trace a path from u and determine the number of possible alternate paths from each intermediate node visited. When we reach a node v on the current path from u, there are at most $(b-1)$ different possible branchings from v to alternate paths. Some of the branches may not lead to valid alternate paths as they go to nodes already visited along the current path (this is equivalent to bumping ancestor nets in the DFS process of Conv-T-DAG), and thus $(b-1)$ is an upper bound on the number of branches at v leading to valid alternate paths. Furthermore, in the worst case, at most L nodes are visited in each path in the current exploration of paths from u. Thus the complexity of Conv-T-DAG is $O((b-1)^L)$; since

```
Algorithm Route-With-B&R(N) /*assign tracks to nets in N*/
Begin
   The overlap graph OG = ∅;
   for each n_i ∈ N
      Insert n_i in the OG and update neighbor nets' TCs;
      Get the track T_k for which n_i has the least TC;
      Assign n_i to track T_k;
      Let S be the set of the nets bumped on T_k by n_i,
      for each n_c ∈ S do
         result = Conv-T-DAG(OG, n_c^{T_k});
         if (result == fail) then
            increase track number, put n_i on new track;
            restore track assignments of all bumped nets
            to that prior to n_i's assignment to T_k;
         endif
      endfor
   endfor
End.
```

Figure 5: *Pseudo code for detailed routing that uses the B&R algorithm.*

$L = O(m)$, this complexity becomes $O((b-1)^m)$. Note that this complexity does not take into account the speedup methods discussed in Sec. 4 that we incorporate into the DFS process. Our empirical results shown in Sec. 5.2 suggest that the average-case complexity of the fast version of `Conv-T-DAG` is linear (it actually shows that the average-case complexity of the detailed router `Route-With-B&R` discussed in Sec. 3.3 that calls the fast version of `Conv-T-DAG` m times is at most quadratic in m). This also shows that the speedup methods developed here are very effective in pruning fruitless search spaces during DFS in `Conv-T-DAG`.

3.3 Applying B&R to Complete Detailed Routing

In Fig. 5, the pseudo code `Route-With-B&R` for detail routing a given set of nets using the B&R method is given. For the next net n_i to be routed in the channels designated by the global router, a track T_k is chosen for which n_i's TC is the least. If n_i does not bump any net in T_k, we are done. Otherwise, a B&R solution is obtained for every O-net n_c bumped by n_i on T_k by calling `Conv-T-DAG`. If such a solution does not exist for some O-net n_c, the number of tracks in the FPGA is increased by one, n_i placed on the new track and the track positions of all other nets are restored.

The next theorem establishes the optimality of `Route-With-B&R`.

Theorem 2 `Route-With-B&R` *obtains a final detailed routing solution with the minimum number of tracks irrespective of the order in which the nets are routed.*

Proof Sketch: If a final detailed routing solution that includes all nets in the minimum number t of tracks exists, it is always possible to arrive at this solution from any partial configuration of routed nets and unrouted new ones by placing the new nets in their final track positions, and by moving all existing nets from their current track positions to the final ones. These moves essentially define a forest of converging T-DAGs (each tree rooted at the O-nets bumped by the route of the new nets), and will be found one by one by `Conv-T-DAG` irrespective of the order in which it is invoked for each O-net, as from Theorem 1 it correctly explores all feasible T-DAG traversals at every stage of the routing configuration. □

For complete detailed routing, the time-efficiency of the DFS algorithm for B&R `Conv-T-DAG` (Fig. 4) becomes critical as many more prior routed nets are bumped compared to an incremental routing application, thus leading to extensive DAG searches. The crux of practically applying this optimal algorithm to the complete routing problem for large FPGA circuits is to determine significant search-space prunings that will not sacrifice optimality, as well as to develop much better DFS ordering heuristics than those given by the 1st-level TC functions of Sec. 3.2. These speedup methods are discussed next.

4 Optimality-Preserving Speedup Methods

We describe here our search-space pruning and lookahead DFS ordering techniques that help us to reduce the search time for the B&R DFS search process of Fig. 4 by a few orders of magnitude.

4.1 Learning-Based Search Space Pruning

We first give some useful definitions.

Ancestor net (AN): Figure 6a shows a search space P rooted at net B and preceded by search path τ_1. An *AN* of search space P along a search path leading to the root net B of P (τ_1 in the e.g. of Fig. 6a) is a net preceding B in the search path. In the figure, A, C, D, K are ANs of search space P and of its root net B.

Obstacle ancestor net (OAN): An *OAN* of a search space P rooted at net B is an AN of P that is a neighbor on the OG of at least one net in P. OANs are so called because they do not allow nets in P overlapping them to transit to tracks in which they currently lie (in the context of the B&R algorithm of Fig. 4 which do not allow cycles in in the T-DAG search process) thus becoming "obstacles" to the movement of nets in P. In Fig. 6a, nets A, D, K are OANs of search space P.

Regular ancestor net (RAN): An ancestor net of a search space that is not its OAN is termed a *RAN* of the search space. In Fig. 6a, net C is a RAN of P.

Net vector (NV): A *net vector* is a two-tuple $(NS; T_j)$, where NS is a set of nets $\{n_1, n_2, \ldots, n_k\}$ that all lie on track T_j (without overlapping each other, i.e., no two nets in NS are neighbors in the OG).

Net pattern (NP): A *net pattern* is a set of NVs such that that no two NVs have the same track.

Ancestor pattern (AP): An *ancestor pattern* of a search space P rooted at net B on path τ is a NP $\{NV_1, NV_2, \ldots, NV_l\}$ such that each net in NV_i, $1 \le i \le l$ is an AN of P and each AN of P belongs to some NV_j, $1 \le j \le l$. For e.g., in Fig. 6a, for search space P rooted at B on path τ_1, the AP is $AP_1 = \{(\{A,D\}; T_0), (\{C\}; T_1), (\{K\}; T_2)\}$, while in Fig. 6b, for search space Q rooted at B on path τ_2, the AP is $AP_2 = \{((A,D); T_3), ((K,X); T_1), (C; T_0), (Z; T_2)\}$.

Obstacle pattern (OP): An *obstacle pattern* of a search space P rooted at net B on path τ is a NP $\{NV_1, NV_2, \ldots, NV_l\}$ such that each net in NV_i, $1 \le i \le l$ is an OAN of P and each OAN of P belongs to some NV_j, $1 \le j \le l$. In Fig. 6a, the OP for search space P is $OP_1 = \{(\{A,D\}; T_0), (\{K\}; T_2)\}$, while in Fig. 6b, the OP for search space Q is $OP_2 = \{((A,D); T_3), ((K,X); T_1), (Z; T_2)\}$.

Net pattern subset: Net pattern NP_1 is said to be a *subset* (denoted $\subseteq$) of net pattern NP_2 if each NV in NP_1 is a subset of some NV in NP_2. For e.g., if $NP_1 = \{NV_1', NV_2', NV_3'\}$ and $NP_2 = \{NV_1, NV_2, NV_3, NV_4, NV_5\}$, and if $NV_1' \subseteq NV_1$ and $NV_2' \subseteq NV_2$ and $NV_3' \subseteq NV_3$, then $NP_1 \subseteq NP_2$. Referring to Fig. 6c, $OP_3 = \{(\{A,D\}; T_3), (\{K\}; T_1)\} \subset OP_2 = \{(\{A,D\}; T_3), (\{K,X\}; T_1), (\{Z\}; T_2)\}$.

Obstacle pattern isomorphism : Obstacle pattern OP_2 is said to be *isomorphic* to obstacle pattern OP_1, if $\exists$ a one-to-one and onto function $f : \{0,1,2,\ldots,t-1\} \to \{0,1,2,\ldots,t-1\}$ where t is the number of tracks, such that if $(NS_i; T_i) \in OP_1$ then $(NS_i; T_{f(i)}) \in OP_2$. In other words, if all the nets in net set NS_i are on the track T_i in OP_1 then all the nets in net set NS_i are on the track $T_{f(i)}$ in OP_2. f is termed the *isomorphism function* from OP_1 to OP_2. In Fig. 6c, $OP_3 = \{(\{A,D\}; T_3), (\{K\}; T_1)\}$ is isomorphic to $OP_1 = \{(\{A,D\}; T_0), (\{K\}; T_2)\}$.

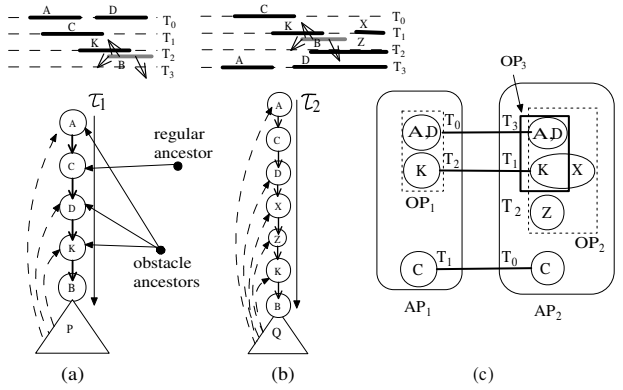


Figure 6: *Net patterns and search paths for a depth-first search. Ancestor nets of the current search space are shown dark in the net pattern drawings. (a) Search space P rooted at B on path τ_1. (b) Search space Q rooted at B on path τ_2. (c) AP_1 is the ancestor pattern of search space P and $OP_1 \subset AP_1$ is the obstacle pattern of search space P. AP_2 is the AP of search space Q and $OP_2 \subset AP_2$ is the obstacle pattern of search space Q. Also $OP_3 \subset OP_2$ and OP_3 is isomorphic to OP_1.*

The learning-based search-space pruning method is now described in terms of the following two fundamental results.

Lemma 1 *Suppose that in a search path τ_1, net B is bumped and there does not exist a solution for this bumping (i.e., the search space P rooted at B on τ_1 is not a converging T- DAG), then if in another search path τ_2 (that may or may not have a common subpath with τ_1), B is again bumped and the ancestor pattern for the search space Q rooted at B on τ_2 is a superset of the obstacle pattern of P, then there does not exist a solution for bumping B in τ_2.*

Proof: In Fig. 6, assume that there is no solution on the path τ_1 when net B is bumped by net K. Let OP_1 be obstacle pattern for search space P on τ_1, and AP_2 is the ancestor pattern for search space Q on τ_2. Some subset P_1 of P can be OANs of B on τ_2 , another subset P_2 of P can be RANs of B on τ_2, and yet another subset P_3 of P can be a subset of Q. Hence P=$P_1 \cup P_2 \cup P_3$. If there exists a solution for bumping B on τ_2, then there exists a valid track position for each net in P_3 (including B). Since the track positions available for subset P_3 of P in τ_2 is a subset of track positions available for P_3 in τ_1 (because $OP_1 \subseteq AP_2$), there exists the same track positions for each $n_i \in P_3$ in τ_1. Further there also exists in τ_1 the same track positions as in τ_2 for the subsets P_1 and P_2 of P in τ_1 (again because $OP_1 \subseteq AP_2$). Hence there is a solution for bumping B in τ_1 and we reach a contradiction. Thus there is no solution to bumping B on τ_2. □

Theorem 3 *Suppose that on a search path τ_1 there does not exist a solution for bumping net B and OP_1 is the obstacle pattern for the search space P rooted at B on path τ_1. Then, if on another search path τ_2 net B is bumped again with an ancestor pattern AP_2, and if there exists a net pattern $OP_2 \subseteq AP_2$ which is isomorphic to OP_1, then there does not exist a solution to bumping net B on τ_2.*

Proof: Assume that there is a solution on path τ_2 when net B is bumped. Let f be the isomorphism $OP_2 \rightarrow OP_1$. If for the solution (i.e., final track positions of all nets) to the bumping of B on τ_2, we swap all the nets on track T_i with all nets on track T_{f_i}, we translate OP_2 to OP_1. The above swappings give us valid positions for all nets including B. Considering only the nets of AP_2 after the swappings, their positions are the same as those on a search path τ_3 in which B is bumped with an ancestor pattern AP_3 that is isomorphic to AP_2 with isomorphism function f. Since $OP_2 \subseteq AP_2$ has been translated to OP_1 after the swapping, $OP_1 \subseteq AP_3$. Further, since the swapping has given us valid positions for every net, we have a solution to bumping B in search path τ_3 when OP_1 is a subset of the AP AP_3 of the search space rooted at B on τ_3. But this contradicts Lemma 1. Hence there can be no solution to bumping B on τ_2. □

During the DFS process of algorithm `Conv-T-DAG` (Fig. 4), if we fail on bumping some net B, we note the corresponding OP and store it as an OP for B. As long as we have not increased the number of tracks in which we are trying to find a routing solution, when we bump B again in another DFS process, we have a suitable data structure that can reasonably quickly detect if any subset of the current AP of B is isomorphic to any stored OP of B (there could be multiple, non-isomorphic OPs of B discovered during various DFS searches). If an isomorphic match is found, then from Theorem 3 we know that the bumping of B in the current search path will not give us any solution and we abandon exploring the search space rooted at B, thus saving a significant amount of search time. Subsequently, B's parent tries a different track transition and the search continues along a different path.

4.2 Clique-Based Search Space Pruning

This method dynamically determines the presence of cliques in the OG among the longer nets, which gives an indication of the minimum number of distinct tracks that would be required for successful routing of all the nets in the clique. We define the term *common unusable track (CUT)* as follows: Track T_j is a *CUT* for clique $\mathcal{C}$ in the current search path τ, if each net in $\mathcal{C}$ is adjacent to at least one *AN* of τ on track T_j. For each clique we dynamically maintain the number of CUTs. This combined with the clique size can help prune unfeasible solution spaces as specified in the next theorem.

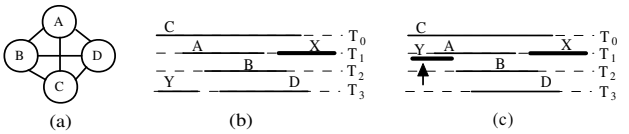
Theorem 4 *Suppose that on search path τ, a net A in a clique $\mathcal{C}$ of size k is bumped. Let t be the current number of tracks and m be the number of CUTs for clique $\mathcal{C}$, after net A is bumped. If $(k+m) > t$ then there is no solution for bumping net A.*

Proof: The nets in clique $\mathcal{C}$ needs k distinct non-CUT tracks to be routed on. Since there are m CUTs, at least $k+m$ tracks are needed for a feasible routing of nets of $\mathcal{C}$ after A is bumped. As the total number of tracks $t < (k+m)$, there is no such routing possible, and hence there is no solution to bumping A in the current search path τ. □

Fig. 7 shows an e.g. of the situation described in the above proof.

4.3 Lookahead TC Functions

As mentioned in Sec. 3.2, the 1st-level TC functions are very useful in pursuing search paths that are more likely to be successful before others that are less likely to yield solutions. However, the 1st-level TCs can still be misleading in some cases. For e.g., let us consider transitions $n_1^{T_0 \rightarrow T_1}$ and $n_1^{T_0 \rightarrow T_2}$ for net n_1, which is on track T_0. Suppose that in the $T_0 \rightarrow T_1$ transition, n_1 bumps a net n_2 of length 7 and in the $T_0 \rightarrow T_2$ transition, n_1 bumps a net n_3 of length 4. The 1st-level TCs will favor pursuing the $T_0 \rightarrow T_2$ transition before trying the $T_0 \rightarrow T_1$ one (if the former cannot find a solution) since the 1st-level TCs are based mainly on the lengths of the bumped nets. However, not counting the single channel in which n_1 overlaps n_2, suppose that n_2 goes through the other 6 channels that are either empty except for n_2 or very sparsely occupied by short nets. Conversely, suppose that the other 3 channels (besides the one in which n_1 and n_3 overlap) occupied by n_3 are all full and occupied by very long nets. It is clear that it will be much more difficult to find a solution in which n_3 is moved from its current track position (due to $n_1^{T_0 \rightarrow T_2}$) to some other track where it is guaranteed to bump long nets, compared to one in which n_2



(a) (b) (c)

Figure 7: *(a) A clique C of size $k = 4$. (b) Track position of each net of C and of an AN X of the current search path. (c) Net A is bumped by net Y causing track T_1 to become a CUT for C. Since the number of tracks $t = 4$ and number of CUTs $m = 1$, the bumping of A will not lead to a solution as can also be seen by inspection of the net routes.*

Ckt Name	# Nets	Opt. Tracks	Ckt Name	# Nets	Opt. Tracks
C499	147	7	vda	399	10
mm9a	205	6	frg2	558	6
alu2	236	7	apex6	575	6
s1	248	7	ex4p	586	6
s1423	272	6	mm30a	651	6
t481	280	8	misex3c	663	9
sand	285	7	ex5p	1072	15
mm9b	292	6	tseng	1248	7
planet	307	6	misex3	1658	12
planet1	308	7	alu4	1748	12
x4	310	6	diffeq	1786	8
s1196	325	7	des	1847	8
i6	328	4	apex2	2284	13
duke2	328	8	elliptic	4175	11
s1488	340	6	spla	4286	14

Table 1: *Characteristics of benchmark circuits. VPR obtains the optimal tracks mentioned here, and those are also obtained independently by all our ROAD routers.*

is moved from its current track position (due to $n_1^{T_0 \to T_1}$) to some other track where it will either not bump any net or will bump very few short ones. Thus looking ahead to the next transitions of the nets bumped by n_1 gives us more accurate information on which transitions of n_1 to try first. Essentially, for a possible transition $n_1^{T_i \to T_j}$, this lookahead information can be captured as some function of all possible 1st-level TCs of each net bumped by n_1 on track T_j. We choose the min function for this purpose as it reflects the (1st-level) cost of the transition of a net bumped by n_1 that is most likely to lead to a solution. The 2nd-level (or lookahead) TC function is then obtained by substituting in the 1st-level TC function of $n_1^{T_i \to T_j}$, the above min function for n_k in place of the length $l(n_k)$, for each net n_k bumped by n_1. Since we have two possible first level TC functions (*sum, sqrt*), we get the following 4 possible 2nd-level TC functions (for the 4 combinations of the "inner" and "outer" 1st-level TC functions used to form the 2nd-level TCs):

(1-2) $TC_2^{sum-x}(n_i^{T_j \to T_k}) = \sum_{n_j \in adj^{T_k}(n_i)} \min_{\forall T_l}(TC_1^x(n_j^{T_k \to T_l}))$

(3-4) $TC_2^{sqrt-x}(n_i^{T_j \to T_k}) = \frac{\sum_{n_j \in adj^{T_k}(n_i)} \min_{\forall T_l}(TC_1^x(n_j^{T_k \to T_l}))}{\sqrt{|adj^{T_k}(n_i)|}}$

where x is either *sqrt* or *sum*.

In particular, using the 2nd-level TC function with the *sum-sqrt* combination yields a speedup factor of 83 over using the best 1st-level TC function *sqrt* (see Table 2).

5 Experimental Results

To test the efficacy of the various speedup techniques of Sec. 4, we started from the basic algorithm `Route-With-B&R` (Figure 5) and added different speedup methods to create the next version until all methods were incorporated. We term the general routing methodology that uses `Routing-With-B&R` and the various speedup methods as *ROAD* (bump&Refit based OptimAl Detailed router). We have the following three versions of ROAD:

- ROAD^{-1}: Basic `Route-With-B&R` using 1st-level TC function (Sec. 3.2) in `Conv-T-DAG`.
- ROAD^0: The 1st-level TC function of ROAD^{-1} is replaced with a 2nd-level TC function *sum-sqrt* (see Sec. 4.3).
- ROAD: Learning-based and clique-based search space pruning methods are added to ROAD^0.

We ran all versions of ROAD as well as VPR on 550 MHz Pentium III Linux workstations. Table 1 shows the benchmark circuits that we used in our experiments. The number of nets for the circuits ranges from 147 to 4286.

5.1 Extracting VPR's Global Routing Topology

Our goal in this paper is to develop an order-impervious pure detailed router (i.e, one that performs only track assignments) that, given a global routing topology, uses the B&R paradigm to obtain a solution that is optimal in the number of tracks used. To this end, we tested all versions of ROAD (referred to collectively as ROAD whenever we are not distinguishing between the different versions) on the benchmark circuits of Table 1 by extracting the global routing topologies from the results of VPR's flat routing (global and detail routing performed in an integrated manner) [13] for these circuits, and discarding all track assignment information. As it turns out, for each circuit, VPR's route causes at least one channel in the FPGA to be fully occupied. This means that if VPR returns a solution with t-tracks, then for the corresponding global topology of the nets, the optimal track assignment should also yield a t-track solution. We ran ROAD with different net orderings and always obtained the detailed routing solutions in the optimal number of tracks specified in Table 1. Note that VPR's overall solution may not be optimal because, say, it may not be inserting hubs in the optimal matter. ROAD is, however, constrained by the hub-based topology yielded by VPR. Thus for the given net topologies, ROAD performs optimally as it is theoretically supposed to do.

5.2 Internal Comparisons

Table 2 shows runtimes for various ROAD versions. For some of the circuits ROAD^{-1} could not complete the routings in a reasonable time (3 hr wall clock time). The runtimes show that ROAD^0 is 83 times faster than ROAD^{-1} while ROAD is 604 times faster than ROAD^{-1} for the circuits that the latter could route within the above time limit. Table 2 also shows that ROAD is 61 times faster than ROAD^0. This gives an extrapolated speedup of 61x83=5763 for ROAD over ROAD^0 for the larger circuits in Table 1. As can be seen, runtimes are not reported for ROAD^0 for all circuits of Table 1 due to it exceeding our prespecified time limit. However, ROAD is able to obtain routings for all circuits in Table 1 and these are reported in Table 3 (discussed shortly).

Figure 8 shows the plot of ROAD run-time in secs versus the number of nets (in logarithmic scale) across the circuits of Table 1, as well as the best linear and quadratic curve fits to this data. As shown in the figure, the quadratic function is a better fit to ROAD run-time. However, note that the coefficient of the second order term in the quadratic function is very small ($3 \cdot 10^{-5}$), and thus when the number of nets is not very large (e.g., < 4000), this term does not have any appreciable effect and the function is almost linear. When the number of nets increases, say, beyond 4000, the 2nd-order term dominates and the function is quadratic. Thus for small- to medium-size circuits, the empirical average-case time complexity of the ROAD is $\Theta(m)$, while for large circuits it appears to be $\Theta(m^2)$, where m is the number of nets in a circuit.

5.3 Comparisons to VPR

To get an idea about how much time VPR spends for detailed routing and thus compare it fairly to ROAD's runtime, for each

Circuit Name	Time in secs		
	ROAD^{-1}	ROAD^{0}	ROAD
mm9a	3299.22	130.03	1.27
alu2	4015.23	1.44	1.41
s1	3163.87	1.40	1.14
s1423	537.34	4.65	2.56
sand	131.76	4.90	2.08
planet	2010.56	2.29	2.05
planet1	1.06	2.24	2.37
x4	18.06	3.41	2.72
i6	4.60	4.56	3.93
s1488	12.21	2.97	2.31
C499	NC	51.52	0.54
t481	NC	37.80	1.80
mm9b	NC	10.05	2.45
s1196	NC	87.70	3.10
duke2	NC	238.20	2.11
vda	NC	1670.47	4.99
Total (10 ckt)	**13193.91**	**157.89**	**21.84**
Speedup over ROAD^{-1}		83.56	604.12
Total (16 ckt)		**2253.63**	**36.83**
Speedup over ROAD^{0}			61.19

Table 2: *The first row of totals show the runtimes for ROAD^{-1}, ROAD^{0} and ROAD for 10 circuits. The second row of totals show the runtimes for ROAD^{0} and ROAD for all circuits (ROAD^{-1} could not get results for these circuits in a reasonable time –NC means not completed).*

circuit in Table 1, we obtained VPR's global routing time and subtracted it from VPR's global + detail (flat) routing time (it is not possible to run VPR in detailed routing mode only). Table 3 shows these estimated times for each circuit. For most of the circuits, ROAD has significantly smaller run-times. For the circuits in Table 3, the total time for ROAD is 2202.64 sec, while for VPR it is 4098.59 sec; ROAD thus achieves a speedup of almost a factor of two over VPR. Note again that both routers yield the same number of tracks for each circuit (see Sec. 5.1).

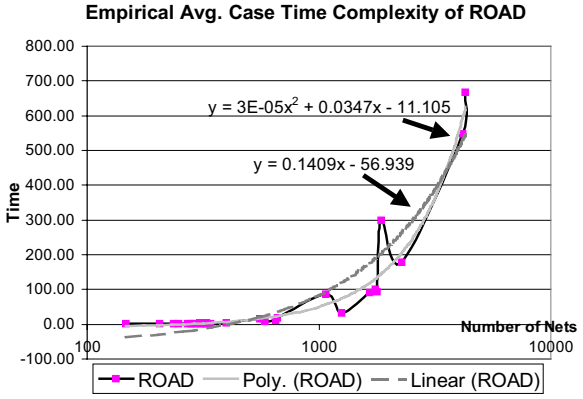


Figure 8: *ROAD run-time in secs on the y-axis versus the number of nets (in logarithmic scale) on the x-axis, and the best linear and quadratic curve fits to this data. The quadratic curve fits ROAD data the best, albeit with a very small coefficient for the 2nd-order term in the function.*

6 Conclusions

We presented a new approach called ROAD to detailed routing in FPGAs that uses the bump-and-refit (B&R) paradigm. It thereby overcomes the well-known net ordering problem of detailed routing and obtains routings in the minimum number of tracks. This is an important advancement in detailed routing technology for current and future FPGA circuits that are generally interconnect dominated. An early basic B&R algorithm searches

Ckt Name	Time (sec.)		Speedup	Ckt Name	Time (sec.)		Speedup
	VPR	ROAD	over VPR		VPR	ROAD	over VPR
C499	3.42	0.54	6.33	vda	34.13	4.99	6.84
mm9a	5.4	1.27	4.25	frg2	55.88	15.12	3.7
alu2	8.54	1.41	6.06	apex6	31.91	15.7	2.03
s1	3.03	1.14	2.66	ex4p	20.35	9.72	2.09
s1423	2.85	2.56	1.11	mm30a	21.91	11.76	1.86
t481	8.41	1.8	4.67	misex3c	29.07	17.11	1.70
sand	5.37	2.08	2.58	ex5p	647.72	87.7	7.39
mm9b	9.1	2.45	3.71	tseng	33.01	31.2	1.06
planet	10.95	2.05	5.34	misex3	260.89	90.25	2.89
planet1	4.6	2.37	1.94	alu4	218.82	101.57	2.15
x4	13.95	2.72	5.13	diffeq	99.47	93.2	1.07
s1196	5.29	3.10	1.71	des	301.93	299.3	1.01
i6	6.43	3.93	1.64	apex2	404.72	178.2	2.27
duke2	10.68	2.11	5.06	elliptic	548.47	546.89	1.00
s1488	12.88	2.31	5.58	spla	1279.41	668.09	1.92
Total for all circuits					**4098.59**	**2202.64**	**1.86**

Table 3: *Comparison of ROAD and VPR (detailed routing) runtimes.*

significant portions of the routing search space, and thus tends to be quite slow when used in a complete detailed router like ROAD (B&R was initially proposed for incremental routing where the search space is significantly smaller) for medium to large circuits. We thus developed a number of optimality-preserving search-space pruning methods and lookahead transition cost functions that yield speedups of a few orders of magnitude. Furthermore, ROAD is almost twice as fast as the estimated detailed routing phase of VPR. The relevant point here is not so much the numerical value of the speedup we obtain over VPR, but that we have been able to develop an optimal routing algorithm that has reasonable runtimes that compares favorably with a non-optimal (though effective) router. Furthermore, for very large FPGA circuits, the flat routing approach of VPR may become very time-expensive, thus calling for a two-stage global followed by detailed routing approach for such circuits. In that case, ROAD would be well positioned to be a high-quality and time-efficient detailed routing phase for such a two-stage router.

References

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

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

[3] E. Rosenberg, "New Iterative Supply/Demand Router with Rip-up and Reroute Strategy", *DAC 1987*.

[4] A. Hetzel, "A Sequential Detailed router for Huge Grid Graphs", *In the Proc. Design Auto. & Test in Europe*, 1998.

[5] H. Shin and A.S. Vincentelli, "A Rip-up and Reroute Detailed Router", *ICCAD* , pp. 2-5, 1986.

[6] H. Shirota et al., "A New Rip-up and Reroute Algorithm for Very Large Scale Gate Arrays", *IEEE CICC 1996*

[7] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-Based Performance-Driven Router for FPGAs", *In Proc. of ACM Symp. on FPGAs*, pp. 111-117, Feb 1995.

[8] N.Mani and N.H.Quach., "Heuristics in the Routing Algorithm for Circuit Layout Design", *IEEE Proc. CDT*, March 2000.

[9] M. Bartholomeus and M. Raith, "New Graph Theo. Appr. to the Selection of Rip-ups", *Euro ASIC91*, pp. 1277-1290, 1991.

[10] F. Mo and A. Tabbara and R. K. Brayton, "A force-directed maze router", *ICCAD*, pp. 404-407, Nov 2001.

[11] P. Groeneveld, "Wire Ordering for Detailed Routing", *IEEE Design and Test of Comp.*, pp. 6-17, 1989.

[12] N. Sherwani and S. Bhingarde and A. Panyam, "Routing in the Third Dimension", *IEEE Press*, 1995.

[13] V. Betz and J. Rose, "VPR: A New Packing, placement and routing Tool for FPGA Research", *Int. Wrkshp. of FPGAs*, London, 1997.