Archer: A History-Based Global Routing Algorithm

Muhammet Mustafa Ozdal and Martin D. F. Wong, *Fellow, IEEE*

***Abstract*—Global routing is an important step in the physical design process. In this paper, we propose a new global routing algorithm *Archer*, which resolves some of the most common problems with the state-of-the-art global routers. It is known that concurrent global routing algorithms are typically too expensive to be applied on today's large designs, which may contain up to a million nets. On the other hand, iterative rip-up and reroute (RNR)-based algorithms are susceptible to getting stuck in local optimal solutions. In this paper, we propose an RNR-based global routing algorithm that guides the routing iterations out of local optima through effective usage of congestion histories. We also focus on the problem of how to enable a smooth tradeoff between seemingly conflicting objectives of overflow and wirelength minimization. Furthermore, we propose a Lagrangian relaxation-based bounded-length min-cost topology improvement algorithm that enables Steiner trees to change dynamically for the purpose of congestion optimization. Our experiments on public benchmarks show the effectiveness of Archer compared to other state-of-the-art global routers.**

***Index Terms*—Congestion optimization, global routing, Lagrangian relaxation, Steiner tree optimization.**

I. Introduction

In the past several years, the circuit densities have been increasing significantly due to increased design sizes and shrinking transistors. The routing problem for integrated circuits is becoming a more and more challenging problem. Furthermore, design for manufacturability rules impose complex constraints on the routing problem, making it even more challenging to devise effective algorithms. Due to the problem complexity, the routing problem is typically solved in two steps: global routing and detailed routing. During global routing, nets are routed on a coarse-grain grid structure with the objective of determining the regions within which each net will eventually be routed. In the detailed routing step, the exact routing solution is determined based on the global routes; so the quality of the final interconnects depends largely on the quality of the global routing solutions. In the current technology, interconnects play a significant role in determining several important metrics such as timing, density, and yield. Hence, it is crucial to have a global routing algorithm that can produce high-quality results.

The global routing algorithms proposed in the literature can be roughly categorized into two categories: concurrent and sequential algorithms. In the first category, global routing is formulated as a combinatorial optimization problem, and approximation algorithms are utilized to find a solution close to the global optimal. However, these methods are typically not scalable to handle today's large design sizes, which may contain several hundred thousands nets. In the second category, several heuristic-based algorithms have been proposed to route the nets based on iterative rip-up and reroute (RNR) techniques. These algorithms have the advantage of being able to handle large design sizes effectively; however, they may lead to suboptimal results due to their heuristic natures. In this paper, we propose a new global routing algorithm that resolves some of the most common problems with RNR-based techniques.

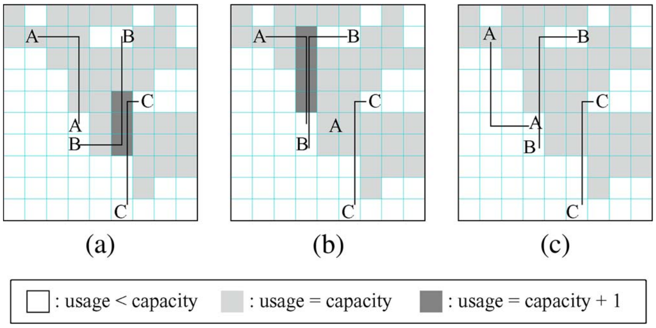


Fig. 1. Routing iterations for three nets. (a) Initial routing with total overflow of three. (b) Total overflow is increased to four after B is rerouted. (c) Congestion-free solution is obtained after rerouting A.

A typical issue with the iterative algorithms is that they are likely to get stuck in local optima during the RNR iterations. Here, net ordering (the order in which nets are routed) can become an important factor that determines the final solution qualities. The state-of-the-art RNR-based global routers are typically greedy in the sense that they always try to minimize an objective function that is a function of congestion cost and wire length. Fig. 1(a) shows an example where three nets are routed on the order: A, B, and C. After the first iteration, nets B and C fail to be routed without overflows; hence, they need to be ripped up and rerouted. In the next iteration, when B is ripped up, it will be routed exactly the same way as in Fig. 1(a), since this is the route that leads to minimum overflow and wirelength. Similar arguments also apply for net C. As a result, the iterative algorithm will be stuck in local optimal and will fail to find the congestion-free solution.[1] Instead, let us consider an alternative metric that will force net B to be rerouted as in Fig. 1(b),

Manuscript received January 11, 2008; revised April 8, 2008. Current version published March 18, 2009. An earlier version of this paper was presented in IEEE/ACM ICCAD in November 2007. This paper was recommended by Associate Editor L. Scheffer.

M. M. Ozdal is with the Strategic CAD Labs, Intel Corporation, Hillsboro, OR 97124 USA (e-mail: mustafa.ozdal@intel.com).

M. D. F. Wong is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2013991

[1] Some routers can reduce the severity of this problem by estimating the demand beforehand, or by using demand threshold in their congestion cost formulation as in FastRoute [22]. However, such approaches do not necessarily solve the problem in the example of Fig. 1. Assume that all unshaded grid cells in this figure have $usage = capacity - 1$. In that case, before net A is routed, the congestion costs of the upper L route and lower L route for net A will be exactly equal, leading to the same problem described in this example.

even though it increases the total overflow. In that case, net A will be congested, and will be forced to be rerouted in the next iteration, as shown in Fig. 1(c). Observe here that the step that leads to the congestion-free solution is the nongreedy step, when net B is rerouted in such a way to increase overflows. In this paper, we propose a global routing framework that allows such nongreedy decisions to avoid getting stuck at local optima. For this purpose, we make use of congestion histories, which are good indicators of how long a particular routing resource has been congested.

Another common problem we focus on in this paper is how to decide which nets to detour and how much to detour for the purpose of overflow minimization. Typically, the iterative global routers use a cost metric that is a weighted sum of congestion cost and wire length. Such a cost metric implicitly assumes that it is possible to quantify the amount of extra wirelength acceptable for each unit of overflow reduction. However, in reality, it is hard to quantify this value, since routability is typically the most important metric, as long as individual net constraints (such as timing constraints) are satisfied. Furthermore, in practice, it is usually the case that detouring the nets prematurely can worsen routability, since more routing resources end up being used due to increased wirelengths. For example, in Fig. 1(a), net C could have been detoured to find a route with less overflow.[2] However, this would not lead to a congestion-free solution. Instead, it would increase the total resource usage in a region where the routing resources are already scarce. In this paper, we propose a robust methodology to determine the nets to be detoured in such a way that there is a smooth tradeoff between seemingly conflicting objectives of overflow and wirelength minimization.

The other important problem we focus on in this paper is the problem of congestion-driven Steiner tree generation during RNR iterations. Here, it is important to let the net topologies dynamically change based on congestion levels. Even though some state-of-the-art global routers allow topology changes during actual routing, the scopes of these methods are usually limited, as will be outlined in Section III. It is possible that minimum length Steiner trees do not always lead to the best topologies in terms of routability. Hence, it is important for the Steiner tree generator to be able to handle the tradeoff between overflow and wirelength minimization objectives. Fig. 2(a) shows an example, where the Steiner points in the min-length Steiner tree are in a congested hotspot. If this net is restricted to this topology, it is possible that congestion will never be able to be resolved since connections need to be done to the Steiner points inside the hotspot. Fig. 2(b) shows another topology, which is suboptimal in terms of wirelength, but avoids the heavily congested hotspot. In this paper, we propose a Lagrangian relaxation (LR)-based bounded-length Steiner topology optimization algorithm to minimize overflows.

The rest of this paper is organized as follows. In Section II, we present the problem formulation, and summarize the related work in Section III. Then, we outline the overview of our global routing framework in Section IV. We present our cost model in Section V-A to resolve congestion effectively based on congestion histories and length bounds. In Section V-B, we propose a routing methodology that enables a smooth tradeoff between overflow and wirelength minimization objectives, as well as scalable CPU times. We then propose an LR-based bounded-length min-cost topology improvement algorithm to minimize overflows in Section VI. Although our models are generic enough to be applied on both 2-D and 3-D grid structures, we discuss some specific issues related to 3-D global routing in Section VII. Finally, we present our experimental results in Section VIII.

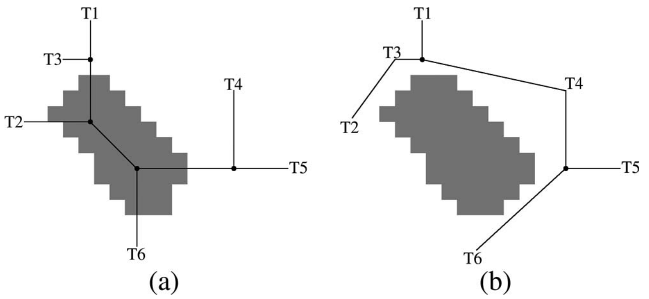


Fig. 2. (a) Min-length Steiner topology for a six-terminal net. Two of the Steiner points (shown as filled circles) are inside a congestion hotspot (shown as the shaded region). (b) Alternative Steiner topology which has larger wirelength, but avoids the hotspots.

II. Problem Formulation

For global routing, the layout is partitioned into rectangular tiles, and a coarse-grain routing grid $\mathcal{G}$ is constructed. Each terminal is assumed to be at the center of the tile that contains it. In this model, each grid tile can be considered as a vertex, and the boundaries between different grid tiles (i.e., grid edges) can be considered as graph edges connecting the neighboring vertices. Here, each grid edge is defined to have a *capacity* value corresponding to the number of available tracks passing through it. The basic objective of global routing is to route all nets on $\mathcal{G}$ such that capacity constraints of edges are satisfied, and the wirelengths are minimized. The number of nets utilizing grid edge e is denoted as *usage* of e. If the usage of e is greater than the capacity of e, then e is defined to be *congested*. The *overflow* of a congested edge e is defined to be the difference between its usage and capacity values. A primary objective of global routing is to minimize the overflow values to ensure routability during detailed routing.

If a 2-D grid model is used [as in Fig. 3(a)], then the routing tracks on every layer are lumped together to compute the edge capacities. On the other hand, a 3-D grid graph can capture the characteristics of different layers more accurately. For example, there can be routing blockages on specific layers, and different layers can have different wire width and spacing requirements based on the technology being used. Although the 3-D grid model can capture the capacity differences of different layers, it requires layer assignment to be performed during global routing. Fig. 3 shows a 3-D grid graph, where each layer has either horizontal or vertical orientation.

The algorithms we propose in this paper are applicable to both 2-D and 3-D models.

[2] Although not shown here, assume that there are uncongested routing resources beyond the rightmost boundary of this figure.

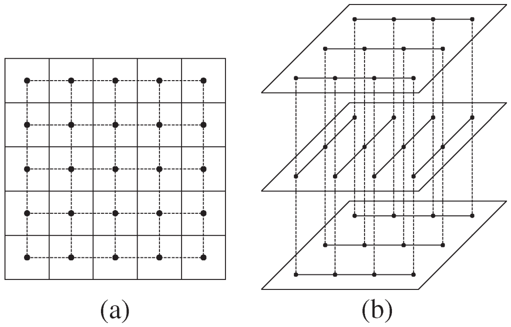


Fig. 3. (a) Two-dimensional routing grid, where the capacities of all layers are lumped into one layer. (b) Three-dimensional routing grid with two horizontal and one vertical layers.

III. Related Work

Global routing has been studied extensively in the literature. In the recent years, it attracted special focus due to increased complexity and increased importance of IC routing. The global routing algorithms can be roughly categorized into two classes: concurrent and sequential algorithms.

Some of the recent concurrent global routing algorithms are [1] and [17]. Typically, the routing problem is formulated as a network flow problem, and approximation algorithms are utilized to solve the mathematical formulation. Although technically sound, these algorithms are typically not scalable for large circuits due to increased complexities.

FastRoute [22], [23] is a recently proposed RNR-based iterative algorithm, and the results reported in [22] and [23] outperform other existing iterative algorithms such as Labyrinth [13] and Chi Dispersion router [10]. However, it suffers from the common problems outlined in Section I. Specifically, the order in which nets are routed can have significant impact on the routing qualities. Moreover, the tradeoff between the objectives of overflow and wirelength minimization is not focused on.

BoxRouter [5] is also a recently proposed global router that outperforms previous algorithms, but gives comparable results to FastRoute. The algorithm starts with routing as many *flat* connections as possible without creating an overflow, followed by prediction of the congested regions. Then, starting from the most congested region, a box is gradually expanded. At each step, the nets that are within the current box are routed using an integer linear programming (ILP) formulation. The nets that cannot be routed without overflow within the current box are rerouted using maze routing. Here, since ILP is an expensive operation, only limited number of nets within the defined box are considered for ILP. Furthermore, only two routing patterns (based on L shapes) are considered for each net, which is a serious limitation of the solution space. Hence, the routing results obtained by ILP are only optimal with respect to a limited number of nets, each net having only two different routing alternatives. Furthermore, the tradeoff between wirelength and overflow is handled through an *ad hoc* methodology, which is based on minimizing a cost function that is a linear function of wirelengths and overflows.

The problem of congestion-driven Steiner tree generation has also been studied extensively in the literature. The Steiner min-max tree algorithm proposed by Chiang *et al.* [3] minimizes the maximum weight of a Steiner tree optimally, while wire length minimization objective is secondary, and handled through heuristics. Here, net ordering can be a serious potential problem. Assume that a set of nets have already been routed, and we are about to route a new net n. Here, the topology of net n will be determined based on the current congestion values. Hence, the wirelength of net n can be increased significantly, which can potentially increase the congestion of the nets to be routed after n. As shown in the example of Fig. 1, sometimes it is important to *negotiate* with the previously routed nets, rather than increasing wirelengths immediately. Another problem is that minimizing the total overflow values is typically more desirable than minimizing the maximum overflow while generating topologies. On the other hand, the problem of generating the Steiner tree with minimum total overflow is NP-complete, and the algorithm proposed in [3] cannot be utilized for that purpose.

There are also mathematical programming models proposed for the congestion-driven Steiner tree generation problem such as [2]. However, these algorithms are typically impractical for large circuits containing several hundred thousands nets. Another common approach utilized by concurrent global routers (such as [1]) is to generate a small set of Steiner topologies in the beginning, and let the network flow algorithm choose the best topology based on congestion. However, it is hard to *guess* the best set of Steiner trees that will lead to lower congestion. As shown in the example of Fig. 2, the Steiner trees that are suboptimal in terms of wirelength can in fact lead to lower congestion values.

FastRoute [22] also utilizes some techniques to generate congestion-driven Steiner trees. The min-length Steiner tree generation algorithm FLUTE [6] is modified to handle congestion costs. Here, each row/column of the Hanan grid is assigned a weight based on congestion map, and FLUTE is applied to minimize these weights. Although practical, this approach has a serious shortcoming of averaging congestion at each row and column. For example, the effect of a hotspot in the middle as in Fig. 2 is averaged out through all rows and columns, and specific location of the congestion is ignored.

In this paper, one of our contributions is to utilize congestion histories to guide the RNR iterations out of local optima, as shown in the example of Fig. 1. The idea of using congestion histories has been used in different contexts such as field-programmable gate array routing [14], [15] and package routing [19], [21]. The idea is to increase the costs of the routing resources that have been congested for several iterations so that only the nets that really need to use these resources end up using them.

An earlier version of our global routing algorithm Archer [20] was presented in International Conference on Computer-Aided Design (ICCAD), 2007, concurrently with two other global routing algorithms: FGR [25] and BoxRouter 2.0 [4]. All three algorithms utilize history costs to resolve congestion, and they report significantly better results than the previously published global routers on the ISPD07 routing benchmarks.

FGR uses Discrete Lagrange Multiplier formulation to model the congestion history costs. Specifically, the congestion cost c_e

of edge e is computed based on the following formula:

$$c_e = b_e + h_e \times p_e. \quad (1)$$

Here, b_e is the base cost of e, h_e is the history cost, and p_e is the congestion penalty term, which is increased exponentially with increasing overflows. Observe that minimization of b_e leads to wirelength minimization, while minimization of $h_e \times p_e$ term leads to congestion spreading. FGR uses A* search algorithm to iteratively reroute congested nets while minimizing this cost function. For multipin nets, FGR uses minimum spanning tree (MST) decompositions to identify point-to-point connections (as opposed to Steiner tree decompositions), since MST topologies enable more flexibility during routing.

BoxRouter 2.0 uses the ILP-based BoxRouter 1.0 [5] algorithm to perform initial routing. After that, the congested nets are iteratively rerouted using a negotiation-based A* search algorithm. The cost of edge e is defined here as the weighted sum of historic and present congestion costs of edge e. The authors also propose heuristic scaling of congestion costs to avoid divergence of iterations. Furthermore, BoxRouter 2.0 uses a sophisticated ILP-based layer assignment algorithm to project the planar routes to multilayer solutions.

While FGR and BoxRouter 2.0 are both based on iterative maze routing, we propose a routing framework in Archer to enable a smooth tradeoff between overflow, wirelength, and CPU time minimization objectives. We also propose an LR-based topology improvement algorithm to minimize congestion while maintaining a maximum length bound. Further experimental comparison between these algorithms is given in Section VIII.

Since the first version of this manuscript was submitted, two new global routing algorithms have been published: NTHU-Route [8] and MaizeRouter [16]. While NTHU-Route uses an improved history-based iterative RNR algorithm, MaizeRouter algorithm is based on two basic operations: extreme edge shifting and edge retraction. Furthermore, MaizeRouter maintains the routing solutions as flat collections of intervals, and operates on entire nets, instead of topology-based point-to-point connections.

IV. Overview of Methodology

The high-level framework for the global router we propose in this paper is outlined in Fig. 4. In the beginning, we use FLUTE [6] to generate a min-length Steiner tree for each net, and we route each topology edge[3] with the objective of minimizing wire lengths. Then, we perform RNR iterations until a congestion-free routing solution is found or until a certain number of iterations have been completed.

In each RNR iteration, we rip-up each congested two-pin connection, and we try to reroute it to minimize the cost function defined in (2), as will be explained in Section V-B. This cost function is based on congestion histories and is expected to lead the connections away from the congested hotspots. In the beginning, we route all the connections using I, L, or Z patterns (Fig. 7), since the CPU requirements for this type of pattern routing is significantly lower than maze routing. If a connection is failed to be routed without congestion in several iterations, then we relax the constraints for that connection. In particular, we start to utilize pattern routing with detours (Fig. 8) or maze routing, while increasing the length bounds gradually. This framework is outlined in Fig. 6, which is explained in detail in Section V-B.

```
ROUTE-ALL-NETS ($\mathcal{G}$: the global routing grid, $\mathcal{N}$: the set of nets)
  for each net $n \in \mathcal{N}$ do
    compute min-length Steiner tree for $n$ using FLUTE [6]
    initial route net $n$ with min length
  while congestion free solution not found do
    in every $K$ iteration do
      for each congested net $n$ do
        improve topology of $n$ (Sec. VI)
    for each congested 2-pin connection $c$ do
      ripup $c$
      reroute $c$ to minimize Equation 2 (Sec. V-A)
      based on congestion history of $c$:
         update constraints of $c$ as in Figure 6 (Sec. V-B)
```

Fig. 4. High-level global routing framework.

In every K iterations, we improve the Steiner topologies of the nets based on the current congestion levels. Note that the initial Steiner trees have been computed to minimize the wire lengths. During RNR iterations, as the congestion hotspots are identified, the topologies of the nets are updated to reduce congestion. It is important here to let the Steiner topologies change dynamically based on current congestion levels, as opposed to using static topologies, which are generated based on congestion estimation in the beginning as in [22]. Note here that topology generation for a net is a more expensive operation than routing two-pin connections. Hence, we perform topology improvement only in certain number of iterations (K). The value of K can be empirically determined based on the tradeoff between solution qualities and runtimes. In our experiments, we have set K to 50.

V. Global Routing Based on Congestion Histories

A. Cost Formulation for Path Computations

Choosing the right cost metric is very important for global routing. A widely used cost function is a weighted sum of congestion cost and wirelength. However, as previously mentioned, increasing wirelengths prematurely to reduce overflows can eventually lead to even higher overflows. To remedy this problem in our algorithm, we maintain a length bound for each net. Initially, the length bound for each connection is set to the Manhattan length of the connection. Then, we gradually increase the length bound of a net if it cannot be routed congestion-free in a certain number of iterations. More details about this operation will be explained later in Section V-B. At any point during RNR iterations, we use the following cost metric (in lexicographic order) to route a two-pin connection, while making sure that the respective length bound is satisfied.

1) Minimize the total congestion history cost of the utilized edges.

[3] Each topology edge of a net will be denoted as *connection* throughout this paper.

2) Minimize the total wire length.
3) Minimize the total usage values of the utilized edges.

Note that we will always choose the route that has lower congestion history cost, while wire length will only be a tie breaker. At first glance, it may seem that this cost metric may lead to unnecessarily increased wire lengths. However, in contrast, this approach increases the wire lengths only when it is really necessary. The reason is that only the connections that could not be routed without congestion in a certain number of iterations are allowed to have length bounds that are greater than the minimum length.

Similar to the formulation in Pathfinder [15], congestion history cost of grid edge e in iteration k is defined as follows:

$$\text{cost}(e) = \left(1 + \alpha . h_e^k\right) \times \text{overflow}(e). \qquad (2)$$

Here, h_e^k represents the history cost for edge e in iteration k, and it reflects for how long edge e has been congested. It is computed as follows:

$$h_e^k = \begin{cases} h_e^{k-1} & \text{if edge } e \text{ is congestion free in iteration } k \\ h_e^{k-1} + k & \text{if edge } e \text{ has nonzero overflow} \\ & \quad \text{in iteration } k. \end{cases} \qquad (3)$$

Therefore, if an edge is congested repeatedly in several iterations, its cost will increase significantly to discourage its usage. Note that this formulation also captures *aging* effect. The edges that were congested only in the earlier iterations will have less costs than the edges that are congested in the later iterations.

In (2), α is the history scale factor that determines the importance of congestion history compared to the actual overflow value. As discussed earlier, it is important for an iterative algorithm not to get stuck at local optima. By introducing the congestion history metric, we enable nongreedy decisions for the purpose of getting closer to the global optimal.

Let us go back to the example of Fig. 1, where the initial routing is demonstrated in part (a). After a few iterations, the history costs of the congested regions in part (a) will become significant enough that net B will be forced to be rerouted to explore alternative routes with smaller history costs. Therefore, it will be rerouted as in part (b), even though the total overflow increases from four to five. This nongreedy step leads to the congestion-free solution of part(c), after net A is rerouted.

Although history costs can be used as an effective metric to avoid getting stuck at local optima, we need to be careful while incorporating it into the cost function, since it may not always directly minimize actual overflow values. To enable smooth transition between these two objectives, we define the history scale factor α in (2) as a function of iteration index, as shown in Fig. 5. The value of α determines the type of optimization being done in the low-level routing iterations. Based on the value of α, we can categorize our iterative algorithm into three main phases: *initiation, negotiation*, and *convergence*.

During the *initiation* phase, the congestion history values are not very accurate; hence, overflow minimization objective is prioritized in (2), by keeping α small. As we perform more routing iterations, the congestion histories start to reflect the congestion hotspots more accurately; hence, α is increased gradually. In the negotiation phase, α has its maximum value, and the congested nets negotiate over the limited routing resources. In this phase, the history costs of congestion hotspots will keep on increasing [due to (2)], and only the nets that really have to utilize these scarce resources will end up using them. In the *convergence* phase, α is reduced gradually to shift the focus back to overflow minimization objective. To enable smooth transition between the objectives of history cost minimization and overflow minimization, we use a continuous function for α, as shown in Fig. 5.

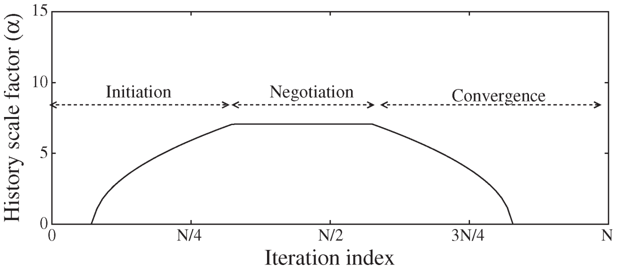


Fig. 5. Plot of history scale factor α as a function of iteration index. A small α value shifts the focus to overflow minimization objective, while a large value increases the importance of congestion history costs.

B. History-Based Routing of Two-Pin Connections

After generating the Steiner topologies for each net, we iteratively RNR the two-pin connections of the nets using the cost function described in Section V-A. Here, there are two important considerations that need to be taken into account.

1) Pattern routing is significantly faster than maze routing. For example, it is reported [22] that 50% of the total runtime is spent on maze routing, even though only 2% of the nets are routed using maze routing. For large designs, containing a few million connections, it will be extremely slow to perform maze routing for each net, and the number of RNR iterations that can be performed will be significantly limited. Hence, it is important to perform maze routing on only the nets that really need it.
2) Increasing wirelengths prematurely to avoid congestion can eventually lead to higher overflows. For example, consider net C in Fig. 1(a), which is routed through a congested region. If net C is detoured just after initial routing to reduce the total overflow, it will end up increasing the usage of routing resources, which are already scarce in that region. Instead, it would have been possible to alleviate congestion through negotiations, as shown in Fig. 1(c). Therefore, it is important to be conservative in terms of allowing detours during RNR iterations.

Based on these considerations, we utilize a set of alternative routing methodologies for different two-pin connections, depending on their congestion histories, as outlined in Fig. 6. Here, each connection is initially allowed to be routed using only pattern routing within length bound $L_{\max}$, where $L_{\max}$ is set to be the Manhattan length of the connection. If a connection is repeatedly congested for several iterations, we gradually increase its length bound $L_{\max}$, and we start to allow maze routing for that connection. Further description of each stage is described below.

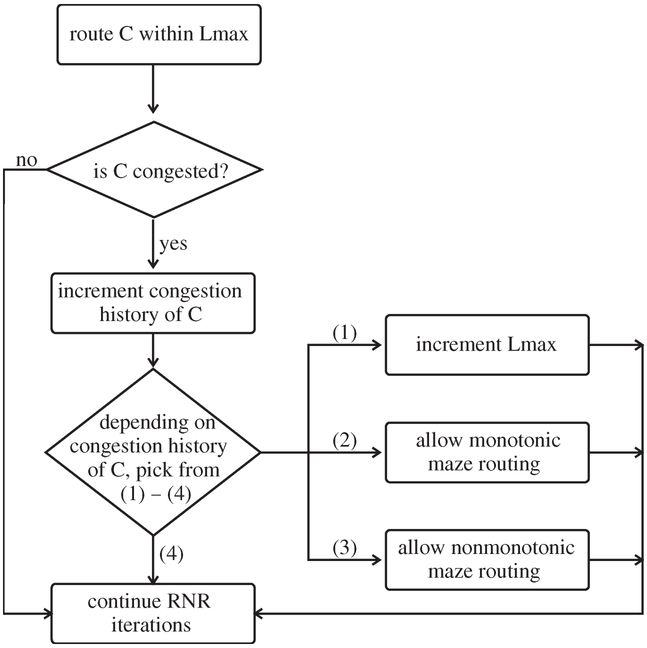

route C within Lmax
is C congested?
no
yes
increment congestion history of C
depending on congestion history of C, pick from (1) – (4)
(1)
increment Lmax
(2)
allow monotonic maze routing
(3)
allow nonmonotonic maze routing
(4)
continue RNR iterations

Fig. 6. Routing methodology for a two-pin connection. Initially, $L_{\max}$ is set to the Manhattan length of the connection, and only pattern routing is allowed.

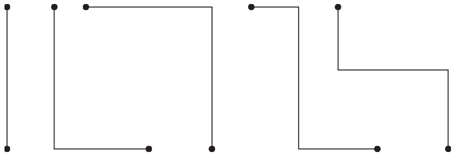


Fig. 7. Min-length pattern routing (I, L, Z routing).

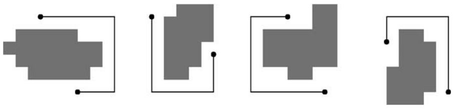


Fig. 8. Pattern routing with detours (U routing).

Min-Length pattern routing (I/L/Z routing): In the beginning of RNR iterations, each connection is restricted to be routed with one of I, L, or Z patterns, which are shown in Fig. 7. It is extremely fast to route the connections this way, and the majority of the nets are expected to be routed during this stage.

Pattern routing with detours (U routing): If a connection cannot be routed in several iterations, its length bound is increased gradually to enable detouring around congested regions, as shown in Fig. 8. Due to the extended solution space, U-routing is not as fast as min-length pattern routing. However, it is still significantly faster than maze routing. The majority of the remaining congested nets are expected to be rerouted during this stage.

Monotonic maze routing: If a connection cannot be routed congestion free after a significant number of iterations, we start allowing monotonic maze routing for that connection, as shown in Fig. 9(a). Although it is significantly slower than pattern routing, maze routing allows more thorough exploration of the solution space. Here, only a small percentage of the nets are expected to be routed with maze routing to limit CPU overhead.

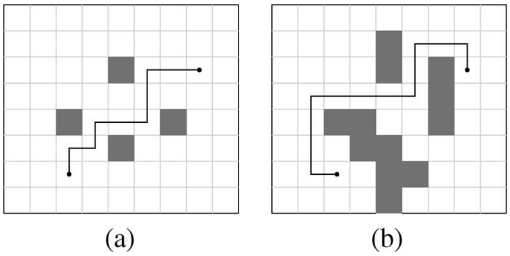


Fig. 9. (a) Monotonic maze routing. (b) Nonmonotonic maze routing.

Nonmonotonic maze routing: If a connection is still congested even after several iterations of monotonic maze routing, we gradually increase its length bound to perform nonmonotonic maze routing. Depending on the length bound of the connection, we define a bounding box around the terminals, and perform nonmonotonic maze routing within this box, as shown in Fig. 9(b). Due to large CPU overheads, only a small percentage of nets that have been repeatedly congested for a very large number of iterations are allowed to be routed using this method.

Observe here that as the size of the solution space explored increases, the runtime requirements of the underlying routing algorithm increase. Furthermore, the algorithms that allow more thorough exploration of the solution space are typically more complicated, making them even slower in practice. For example, monotonic maze routing can be solved in linear time by using a shortest path algorithm for acyclic graphs. Similarly, Z-shaped pattern routing requires at least linear time, since every edge between source and target needs to be queried at least once. However, the number of conditional operations executed in maze routing can be significantly larger than the one in Z-routing, depending on the relative positions of the source and target pins. Hence, this makes monotonic maze routing be slower than Z-routing in practice, even though they have the same asymptotic time complexity.

By following the methodology described in this section, we can obtain a good tradeoff between congestion, wire length, and execution times. The number of iterations needed for each stage can be determined empirically based on the practical requirements. For example, by increasing the number of iterations in which min-length pattern (I/L/Z) routing is utilized, we can perform more aggressive congestion negotiation without wirelength degradations. However, this may lead to increased execution times if detours are absolutely necessary to resolve congestion. On the other hand, reducing the number of pattern routing iterations too much may increase the number of congested nets that need to use maze routing, which may lead to increased execution times. Hence, the determination of the number of iterations for each stage can be done empirically to obtain a good balance between different objectives.

C. Routing Connections of Multipin Nets

We use FLUTE [6] to generate the initial topologies of multipin nets. As outlined in Fig. 4, we optimize these topologies to minimize congestion in every K iterations. Then, we route multipin nets based on their current topologies in each iteration.

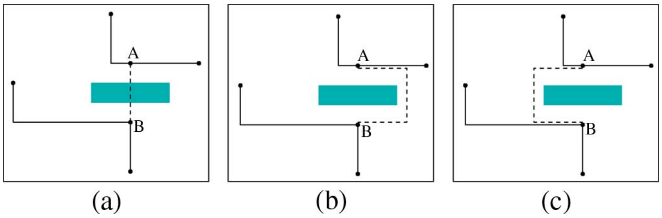


Fig. 10. (a) Connection between pins A and B needs to be routed through a congested region (shaded rectangle). (b), (c) Two U-route alternatives avoiding congestion.

A topology T is defined to have a set of nodes (consisting of net pins and Steiner nodes), and a set of edges connecting these nodes. Each topology edge here is in fact a connection between two nodes, and it can be routed using the methodology proposed in Section V-B. However, special consideration needs to be taken into account to avoid redundant cost computations for the overlapping routing segments of different connections.

Fig. 10(a) shows a partially routed net, and the next connection is to be made between nodes A and B. If a U-route is chosen [as in parts (b) and (c) of the same figure] to avoid the congested region, then we need to consider the overlaps with the existing routes during cost computations. In this example, the solution in part (c) is the preferred one (assuming equal history costs), since the total wirelength of the additional segments is smaller.

This problem can be solved using multisource multisink maze routing as done in [23]. However, in the framework that we propose, not only maze routing but also pattern routing needs to be aware of these overlaps. The reason is that the majority of the nets are expected to be routed by pattern routing, as discussed in Section V-B. For this purpose, we assign zero costs for the routing resources which already have been utilized by the same net, but by other connections. In this way, the overlapping segments will have zero costs, and accurate resource usage will be taken into account during cost computations. Note that similar ideas are utilized in FGR [25] and MaizeRouter [16] to handle overlapping routing segments of multipin nets.

VI. Congestion-Driven Topology Optimization

In this section, we propose an LR-based bounded-length minimum-cost topology improvement algorithm, which is outlined in Fig. 11. Here, the objective is to create a topology for net n that minimizes the objective function defined in (2) while satisfying the length bound $L_{\max}$. Here, $L_{\max}$ is initially set to the length of the original Steiner topology, which is generated with the objective of length minimization. If a net is congested repeatedly for several iterations, the length bound $L_{\max}$ is increased gradually, as described in Section V-B.

The algorithm outlined in Fig. 11 starts with creation of the Hanan grid [11] based on the terminal positions of net n. After the Hanan grid is created, congestion costs are assigned to each edge based on (2). Then, the original net topology T is mapped to this Hanan grid.

It is well known that the problem of finding the minimum cost Steiner tree is NP-complete. Furthermore, enforcing a length bound makes the problem even more difficult. Hence, we use a heuristic-based iterative RNR algorithm to improve the original Steiner tree in terms of congestion costs. Note here that greedily enforcing the input length bound $L_{\max}$ during each RNR iteration is an overconstraint. It is sometimes necessary to allow intermediate topologies of which lengths are greater than $L_{\max}$ to be able to avoid getting stuck at local optima. For this purpose, we propose an LR-based iterative improvement algorithm.

```
IMPROVE-TOPOLOGY ($n$, $T$, $L_{max}$)
  // $n$: the input net
  // $T$: original topology of net $n$
  // $L_{max}$: upper bound for total wire length
  create Hanan grid $\mathcal{G}$ and assign congestion costs to edges
  map original topology to Hanan grid
  initialize Lagrangian multiplier $\lambda$ to 0
  for a fixed number of iterations do
    for each connection $c$ in $T$ do
      rip-up $c$
      reroute $c$ on $\mathcal{G}$ to optimize Equation 5
    update $\lambda$ based on length of $T$
  return best topology $T_{best}$ that satisfies $L_{max}$ bound
```

Fig. 11. LR-based bounded-length min-cost topology improvement algorithm.

LR is a general technique for solving optimization problems with difficult constraints. The main idea is to replace each complicated constraint with a penalty term in the objective function. Specifically, each penalty term is multiplied by a constant called Lagrangian multiplier (LM), and added to the objective function. The Lagrangian problem is the optimization of the new objective function, where difficult constraints have been relaxed and incorporated into the new objective function. If the optimization is a minimization problem, then the solution of the Lagrangian problem is guaranteed to be a lower bound for the original optimization. Therefore, the objective is to find the best LM values such that the optimal value obtained for the Lagrangian problem is as close to the real optimal value as possible. For this purpose, the LM values are updated iteratively (typically using subgradient method) in the high level, while the relaxed Lagrangian problem is solved for the fixed LM values in low-level iterations. Further details about LR can be found in [7] and [9].

The main objective here is to find the best Steiner topology that minimizes the sum of congestion history costs, as defined in (2), under the length constraint of $L_{\max}$

$$\begin{aligned} &\text{minimize} && \sum_{e \in \mathcal{T}} \text{cost}(e) \\ &\text{subject to :} && \sum_{e \in \mathcal{T}} \text{length}(e) \leq L_{\max}. \end{aligned} \tag{4}$$

Here, T is the Steiner topology to be obtained, and e is a Hanan grid edge in T. The cost of edge e is as defined in (2). If we apply LR on this formulation, our objective becomes the minimization of

$$\sum_{e \in \mathcal{T}} \text{cost}(e) + \sum_{e \in \mathcal{T}} \lambda \cdot \text{length}(e). \tag{5}$$

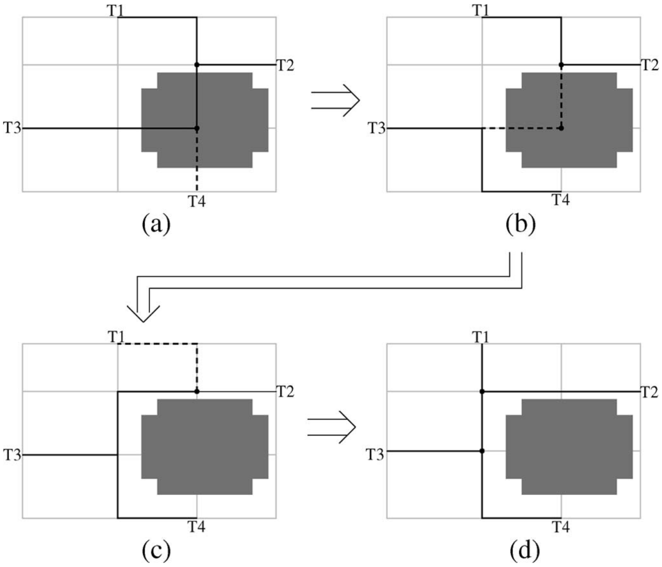


Fig. 12. Example execution of the topology improvement algorithm. The dashed lines represent the connections to be ripped up in the next step. The shaded region represents the congested hotspot.

Here, λ is the LM value corresponding to the length constraint, which is updated after every iteration. Intuitively, if the length of T is greater than $L_{\max}$ after an iteration, then λ is increased to prioritize the length minimization objective, and vice versa. Given λ^k in iteration k, λ^{k+1} for iteration $k+1$ is computed based on the length of T as follows:

$$\lambda^{k+1} = \max(0, \lambda^k + t_k) \times \begin{cases} 1 & \text{if length}(T) > L_{\max} \\ -1 & \text{if length}(T) \le L_{\max}. \end{cases} \tag{6}$$

Here, we use an update schedule similar to subgradient method. Note that t_k is the step size used in subgradient method, and it is updated in each iteration such that it slowly converges to zero. Specifically, we use the convergence condition given by Held *et al.* [12], which states that as $k \to \infty$, it should be the case that $t_k \to 0$ and $\sum_{i=1}^{k} t_i \to \infty$. In particular, we have used $t_k = 1/k^{\alpha}$ in our experiments, where $\alpha < 1$ is a constant to provide a tradeoff between solution quality and convergence speed. In our experiments, we have set α to 0.1.

Based on the objective function given in (5), we iteratively RNR the connections of T to minimize the LR cost. To choose the final topology, we utilize a hill-climbing technique. As previously mentioned, nongreedy steps are important to avoid getting stuck at local optima. The LR cost function enables these nongreedy steps by relaxing the length constraint. At each iteration, the tradeoff between the objectives of congestion cost minimization and length minimization may change depending on the LM value λ. Hence, we keep track of the best topology obtained during these iterations, which has the minimum congestion cost while satisfying the length bound $L_{\max}$. At the end, this best topology is routed in the original routing grid and compared with the cost of the original topology. The new topology is accepted for the net if it ends up reducing the total cost. Otherwise, the original topology is retained.

Fig. 12 shows a sample execution of this algorithm on a simple example. In the original topology [part (a)], there is a Steiner point in the congested hotspot, and there are three connections that need to be routed through the congested region. First, a nongreedy step is performed to obtain the topology in part (b). Observe here that the length of T is increased in this step possibly beyond $L_{\max}$. In the next step, the connection ripped up [the dashed lines in part (b)] is rerouted to reduce congestion while maintaining the same wire length. Finally, another connection is ripped up and rerouted to obtain the final topology of part (d), which avoids passing through the congested hotspot.

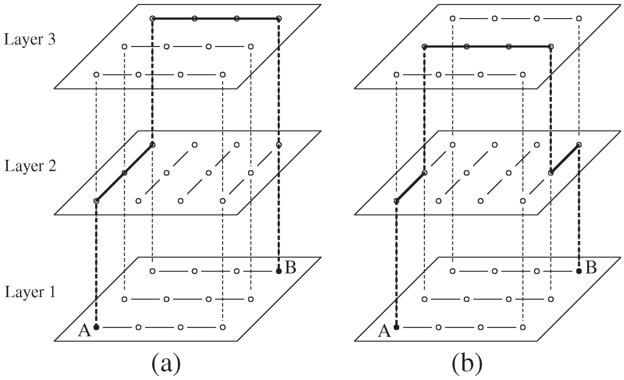


Fig. 13. Pattern routing on a 3-D grid structure between pins A and B. (a) Multilayer L-route. (b) Multilayer Z-route.

VII. Multilayer Global Routing

As discussed earlier, the algorithms proposed in the previous sections are generic enough to be applied on both 2-D and 3-D grid models. In this section, we discuss specific issues related to two alternative methodologies that can be utilized to solve the 3-D global routing problem.

A. Three-Dimensional Path Computations

Since maze routing is inherently a graph-based algorithm, it can be applied on a 3-D grid, as well as a 2-D grid. However, the models and algorithms for pattern routing need to be extended to handle multiple layers.

Fig. 13 shows an L-shaped pattern and a Z-shaped pattern for a 3-D grid structure. Observe here that, the planar routing segments still conform to the specific shape of the pattern; however, each routing segment is allowed to be assigned to a different layer depending on the congestion and via costs. Therefore, the objective of the path computation is to identify the best pattern and the layer assignment that will minimize the total cost.

The graph model used for the L-shaped pattern of Fig. 13(a) is shown in Fig. 14(a). In this specific type of pattern, there is a vertical segment originating from the source pin, and a horizontal segment entering the target pin. To preserve the L-shaped topology, each of these two wire segments need to be assigned to a layer, while vias are allowed only at the end points of the segments. In the graph model of Fig. 14(a), each horizontal/vertical edge (shown as a straight line) represents a possible layer assignment for the corresponding horizontal/vertical segment. On the other hand, the intralayer connections (shown as dashed lines) represent the possible via connections.

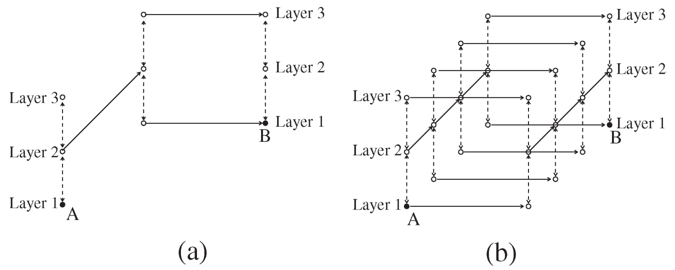


Fig. 14. Graph model for multilayer (a) L-shaped pattern routing and (b) Z-shaped pattern routing.

```
LAYER-ASSIGNMENT ($G_{2D}$, $G_{3D}$, $\mathcal{N}$)
  // map routes of all nets in $\mathcal{N}$ from $G_{2D}$ to $G_{3D}$
  sort the nets in increasing order of semi-perimeter lengths
  for each net $n \in \mathcal{N}$
    $S \leftarrow$ the set of maximal planar segments of $n$ in $G_{2D}$
    for each $s \in S$
      assign $s$ to a layer in $G_{3D}$ obeying overflow constraints
      if failed to assign $s$ to a layer
        add $s$ to the unrouted segments of $n$
  for each net $n \in \mathcal{N}$
    for each unrouted segment $s$ of $\mathcal{N}$
      route $s$ in $G_{3D}$ while obeying:
        the route of $s$ in $G_{2D}$, and
        the overflow constraints in $G_{3D}$
```

Fig. 15. Layer assignment algorithm for projection-based routing.

The cost of these edges are set based on the corresponding cost of the respective connections. Computing the shortest path on this graph model will give us the best layer assignment for this particular L-shaped topology.

A similar model is given in Fig. 14(b) for a Z-shaped topology, which has a vertical segment originating from the source, followed by one horizontal and one vertical segment.[4] Similarly, vias are allowed only at the endpoints of the segments. The shortest path computation on this graph will give us the best route and the best layer assignment for this particular Z-shaped topology.

B. Projection-Based Routing

The second approach to solve the 3-D global routing problem is as follows: 1) project the 3-D grid G_{3D} to the 2-D grid G_{2D}; 2) find the routing solution for G_{2D}; 3) project the solution to G_{3D} through layer assignment. Here, the projection from G_{3D} to G_{2D} is done by simply aggregating the capacity values of the edges on different layers into one edge.

The advantage of this model is the significantly reduced problem complexity. For history-based negotiated congestion algorithm to work effectively, several RNR iterations are needed so that the history costs of the routing resources reflect the real demand accurately. As the number of alternative routing paths for the nets increases, the number of RNR iterations needed to obtain accurate history costs also increases in general. As a result, increasing the problem size typically increases: 1) the runtime requirements of path computations and 2) the number of RNR iterations. Hence, using the planar graph G_{2D} instead of the original graph G_{3D} reduces the problem complexity significantly.

On the other hand, the disadvantage of this approach is the lack of layer awareness during the actual global routing. This may lead to suboptimal results particularly when different layers have different characteristics. Furthermore, the effects of vias is ignored in this model during congestion computations, since routing is done on a planar grid.

Our multilayer experiments are done on ISPD07 benchmarks [18], which are the most challenging publicly available global routing benchmarks today. In these benchmarks, all layers have identical width and spacing constraints, and the congestion impact of the vias are not modeled. Furthermore, the capacity value of an edge in a two-layer circuit is simply the sum of the capacity values of the same edge in the corresponding multilayer circuit. These specifications ensure that the routing solution for G_{2D} can be projected to a solution on G_{3D} without increasing the total overflow of G_{2D}. This property makes the advantages of projection-based routing outweigh its disadvantages on the ISPD07 benchmarks. For fair comparisons with other available results, we chose to implement this technique to route the 3-D circuits.

For this purpose, we have implemented a straightforward layer assignment algorithm to map the results from G_{2D} to G_{3D}, which is given in Fig. 15. Here, the objective is to minimize the number of vias while obeying the 2-D topologies of the routing segments. In the beginning, we sort the nets in increasing order of their semiperimeter lengths. Then, two passes are performed over the planar net segments in G_{2D}. In the first pass, we try to assign each segment s to a layer in G_{3D} such that 1) the number of vias with respect to the assigned segments of net n is minimized, and 2) the capacity constraints of edges in G_{3D} are obeyed. If a segment s cannot be assigned to a layer without violating the capacity constraints, then it is added to the set of unrouted segments. In the second pass, we process these unrouted segments, and we try to route them (not necessarily in a planar way) such that: 1) the number of vias is minimized, 2) the shape of the original 2-D route is preserved, and 3) the capacity constraints of edges in G_{3D} are obeyed.

Compared to an earlier version of Archer [20], the most notable improvement in the algorithm of Fig. 15 is that layer assignment is done in increasing order of segment lengths. When shorter segments are routed before the longer ones, the lower layers tend to be utilized by shorter segments (since lower layers require less vias), while the longer segments are mostly assigned to upper layers. Note that assigning a long planar segment to an upper layer requires less vias then assigning a number of short segments to the upper layer. Hence, the via requirements can be reduced by processing the segments in such a sorted order.

VIII. Experimental Results

We have implemented our algorithms in C++, and performed experiments on a computer with Intel Xeon 3.60-GHz processor and Linux operating system. Note that an earlier version of our global router Archer was presented in ICCAD-07 [20]. There

[4] The 2-D topology for this particular case is shown as the rightmost pattern in Fig. 7.

TABLE I
EXPERIMENTAL RESULTS ON ISPD98 BENCHMARKS

			Archer			FastRoute 2.0			BoxRouter 1.0			Chi Dispersion Router		
Name	# nets	# pins	ovfl	length	cpu (s)	ovfl	length	cpu (s)	ovfl	length	cpu (s)	ovfl	length	cpu (s)
ibm1	11507	44266	0	64389	11	31	68489	1	102	65588	8	189	66005	9
ibm2	18429	78171	0	171805	25	0	178868	1	33	178759	34	64	178892	26
ibm3	21621	75710	0	146770	10	0	150393	1	0	151299	17	10	152392	25
ibm4	26163	89591	0	169977	24	64	175037	2	309	173289	24	465	173241	33
ibm5	27777	124438	0	409761	8	–	–	–	0	409747	50	0	412197	69
ibm6	33354	124299	0	278841	23	0	284935	1	0	282325	33	35	289276	53
ibm7	44394	164369	0	370143	25	0	375185	2	53	378876	51	309	378994	80
ibm8	47944	198180	0	404530	42	0	411703	2	0	415025	93	74	415285	73
ibm9	50393	187872	0	414223	37	3	424949	2	0	418615	64	52	427556	87
ibm10	64227	269000	0	583805	45	0	595622	3	0	593186	95	73	599937	140

TABLE II
RESULTS OF THE CONCURRENTLY PRESENTED ROUTERS ON THE ISPD98 BENCHMARKS

			Archer			FGR			BoxRouter 2.0		
Name	# nets	# pins	ovfl	length	cpu (s)	ovfl	length	cpu (s)	ovfl	length	cpu (s)
ibm1	11507	44266	0	64389	11	0	63332	10	0	66529	4
ibm2	18429	78171	0	171805	25	0	168918	13	0	180053	5
ibm3	21621	75710	0	146770	10	0	146412	5	0	151185	4
ibm4	26163	89591	0	169977	24	0	167101	29	0	176765	27
ibm5	27777	124438	0	409761	8	0	409739	6	-	-	-
ibm6	33354	124299	0	278841	23	0	277608	18	0	288420	8
ibm7	44394	164369	0	370143	25	0	366180	20	0	377072	14
ibm8	47944	198180	0	404530	42	0	404714	18	0	418285	17
ibm9	50393	187872	0	414223	37	0	413053	20	0	431298	17
ibm10	64227	269000	0	583805	45	0	578795	92	0	610680	17

TABLE III
COMPARISON WITH THE RESULTS OF ISPD-07 GLOBAL ROUTING CONTEST

			Archer		FGR		MaizeRouter		BoxRouter 1.9		FastRoute 2.0	
Name	# nets	# pins	ovfl	len+via	ovfl	len+via	ovfl	len+via	ovfl	len+via	ovfl	len+via
adaptec1-2D	219K	940K	0	5766K	0	5580K	0	6226K	0	5884K	122	9047K
adaptec2-2D	260K	1059K	0	5462K	0	5369K	0	5723K	0	5569K	500	8246K
adaptec3-2D	466K	1868K	0	13518K	0	13334K	0	13775K	0	14087K	0	20253K
adaptec4-2D	515K	1904K	0	12632K	0	12605K	0	12845K	0	12875K	0	17080K
adaptec5-2D	867K	3476K	0	16271K	0	15582K	0	17669K	0	16432K	9680	25168K
newblue1-2D	331K	1220K	494	4874K	1218	4751K	1348	5093K	400	5113K	1934	7410K
newblue2-2D	463K	1761K	0	7794K	0	7767K	0	7964K	0	7978K	0	11495K
newblue3-2D	551K	1901K	31928	10959K	36970	10818K	32588	11463K	38976	11164K	34236	15459K
Total			32422	77276	38188	75806	33936	80758	39376	79102	46472	114158
Scaled			1.0	1.0	1.78	0.98	1.04	1.05	1.21	1.02	1.43	1.48

were also two other global routers FGR [25] and BoxRouter 2.0 [4], which were presented concurrently with Archer. In this section, we provide experimental comparison with these concurrently presented routers, as well as the previous global routers.

We have performed two sets of experiments for comparison. First, we have run our global router *Archer* on ISPD98 benchmarks, and compared our results with the previous global routers, as shown in Table I. Observe that Archer successfully resolves the congestion in all circuits, while FastRoute 2.0 [23] and BoxRouter 1.0 [5] fail to find congestion-free routing solutions in several of the circuits. Furthermore, Archer outperforms these routers not only in terms of total overflows, but also in terms of total wirelength, as shown in this table. Table II compares the results of Archer, FGR [25], and BoxRouter 2.0 [4], all of which were presented concurrently. Observe that all three routers successfully route all benchmarks with similar wirelengths, where FGR and BoxRouter 2.0 result in 0.6% smaller, and 3.2% larger wirelengths than Archer, respectively.[5] This necessitates comparison on more challenging benchmarks. Note that the set of parameters used in Archer in this experiment are identical for all circuits, while it is not specified in [25] and [4] whether parameter fine tuning was done for individual benchmarks in the reported FGR and BoxRouter 2.0 results.

Our second set of experiments have been performed on ISPD07 benchmarks, which were released recently, as part of the *ISPD-07 Global Routing Contest* [18]. Each circuit in these benchmarks has a two-layer version, and a multilayer version. As described in Section VII-B, we project the solution of a two-layer circuit to the corresponding multilayer circuit through layer assignment.

We first compare our two-layer results with the best results reported in the *ISPD-07 Global Routing Contest* in Table III. Since CPU time is not a competition metric, execution times are not reported in the contest. Furthermore, using different sets of parameters (and different fine tunings) for different circuits was allowed in the contest. Since overflow minimization is the primary objective of the contest, all algorithms (including Archer) have been tuned to result in the smallest overflow values, while wirelength minimization is the secondary objective. In the contest results, wirelength of a routing solution is

[5] The results of BoxRouter 2.0 are the ones tuned for runtime, as reported in [4]. When tuned for quality, BoxRouter 2.0 achieves 3% smaller wirelength in the expense of about 10× runtime penalty.

TABLE IV
RESULTS OF THE CONCURRENTLY PRESENTED ROUTERS ON THE TWO-LAYER ISPD07 BENCHMARKS

	Archer					FGR					BoxRouter 2.0	
Name	ovfl	length	# via	len+via	cpu (m)	ovfl	length	# via	len+via	cpu (m)	ovfl	len+via
adaptec1-2D	0	3736K	697K	5827K	86	0	3588K	619K	5444K	271	0	5837K
adaptec2-2D	0	3354K	703K	5463K	22	0	3321K	636K	5230K	34	0	5569K
adaptec3-2D	0	9705K	1280K	13545K	49	0	9609K	1160K	13089K	107	0	13796K
adaptec4-2D	0	8922K	1236K	12630K	11	0	9002K	1166K	12500K	11	0	12779K
adaptec5-2D	0	10612K	1880K	16252K	246	0	10279K	1645K	15213K	427	0	16211K
newblue1-2D	682	2452K	797K	4843K	49	514	2415K	776K	4742K	863	400	5113K
newblue2-2D	0	4630K	1054K	7792K	6	0	4681K	990K	7651K	2	0	7868K
newblue3-2D	33394	7566K	1121K	10929K	162	39828	7563K	1120K	10923K	931	38958	11161K
Total	34076	50977K	8768K	77281K	631	40342	50458K	8112K	74792K	2646	39358	78334K
Scaled	1.00	1.00	1.00	1.00	1.00	1.18	0.99	0.93	0.97	4.19	1.16	1.01

TABLE V
RESULTS OF THE CONCURRENTLY PRESENTED ROUTERS ON THE MULTILAYER ISPD07 BENCHMARKS

	Archer					FGR					BoxRouter 2.0	
Name	ovfl	length	# via	len+via	cpu (m)	ovfl	length	# via	len+via	cpu (m)	ovfl	len+via
adaptec1-3D	0	3736K	1960K	9616K	87	0	3637K	1736K	8845K	257	0	9204K
adaptec2-3D	0	3354K	2133K	9754K	23	0	3374K	1871K	8989K	38	0	9428K
adaptec3-3D	0	9705K	3825K	21182K	50	0	9702K	3421K	19966K	146	0	20741K
adaptec4-3D	0	8922K	3508K	19446K	12	0	9128K	3056K	18296K	33	0	18642K
adaptec5-3D	0	10612K	5815K	28056K	247	0	10389K	5203K	25998K	443	0	27041K
newblue1-3D	682	2452K	2383K	9603K	50	514	2415K	2337K	9426K	863	394	9294K
newblue2-3D	0	4630K	3248K	14375K	7	0	4791K	2808K	13216K	6	0	13464K
newblue3-3D	33394	7566K	3366K	17663K	163	39828	7563K	3269K	17371K	899	38958	17244K
Total	34076	50977K	26238K	129695K	639	40342	50999K	23701K	122107K	2685	39352	125058K
Scaled	1.00	1.00	1.00	1.00	1.00	1.18	1.00	0.90	0.94	4.20	1.15	0.96

reported as the actual wirelength plus three times the number of vias. For one-to-one comparison with other routers, the lengths reported in Table III are computed using this metric. As shown in Table III, Archer obtains the lowest total overflow value compared to other global routers. In terms of wirelength, FGR results in 2% less wirelength than Archer. On the other hand, BoxRouter 1.9, MaizeRouter, and FastRoute 2.0 result in 2%, 5%, and 48% more wirelength than Archer, respectively.

We also compare our results with the other global routers presented concurrently with Archer. The results of the two-layer and the multilayer ISPD07 benchmarks are given in Tables IV and V, respectively. Here, the via counts are scaled by three and added to the wirelengths in columns "len + via," in accordance with the ISPD07 contest metric. For BoxRouter 2.0, the individual CPU times are not reported; however, it is stated that newblue3 is routed in more than two days [4]. The CPU times of FGR in these tables are scaled down by 1.67 due to the different platforms used to collect the results of Archer and FGR. This scaling factor between the two platforms was empirically observed by the authors of FGR [24]. Note that the set of parameters used in Archer in these experiments are identical for all circuits, except for *newblue3*, which is assigned an earlier termination condition due to the fact that it is unroutable.[6] It is not specified in [25] and [4] whether parameter fine tuning was done for individual benchmarks in the reported FGR and BoxRouter 2.0 results.

Although most benchmarks are routed overflow-free by all three algorithms, the total overflow values reported by FGR and BoxRouter 2.0 are 18% and 16% larger than Archer, mainly due to the large differences in the results of the unroutable benchmark *newblue3*. On the other hand, FGR and BoxRouter 2.0 result in 3% smaller and 1% larger total lengths (including wires and vias), respectively, compared to Archer on the two-layer benchmarks. For the multilayer benchmarks in Table V, the total lengths of FGR and BoxRouter 2.0 are 6% and 4% smaller than Archer. Note that this difference is mainly because of the via counts, since the planar wirelengths of Archer and FGR are almost identical. We believe that there are two main reasons why the via counts of FGR are smaller than Archer in general. The first reason is that FGR uses flexible spanning-tree topologies, while Archer uses FLUTE [6] to generate Steiner trees, which are optimized for planar wirelengths, but not via counts. It is reported that via counts of FGR have been reduced by about 2% by using spanning trees instead of Steiner trees [25]. The second reason is due to the different cost metrics utilized by Archer and FGR. As discussed in Section V-B, the majority of the nets are expected to be routed with min-length pattern (I/L/Z) routing in the first stage of Archer. Although the planar wirelengths of these patterns are identical, the via counts of different patterns (e.g., L-shaped versus Z-shaped routes) may be different. The cost metric proposed in Section V-A prioritizes congestion costs over wirelength costs. Hence, a Z-shaped route with no congestion will always be preferred over an L-shaped route with congestion despite the increased via usage. Such a cost metric may lead to faster congestion resolution, possibly in the expense of slight increase in the number of vias. Compared to BoxRouter 2.0, the total length reported by Archer is smaller (by 1%) for two-layer benchmarks, but larger (by 4%) for multilayer benchmarks. This can be attributed due to the fact that Archer uses a simple layer assignment algorithm in postprocessing to obtain the multilayer

[6] For *newblue3*, we reduced the maximum number of RNR iterations to avoid rerouting a large number of congested nets repeatedly. We have observed that performing more RNR iterations for *newblue3* leads to increased execution times with little improvement in congestion.

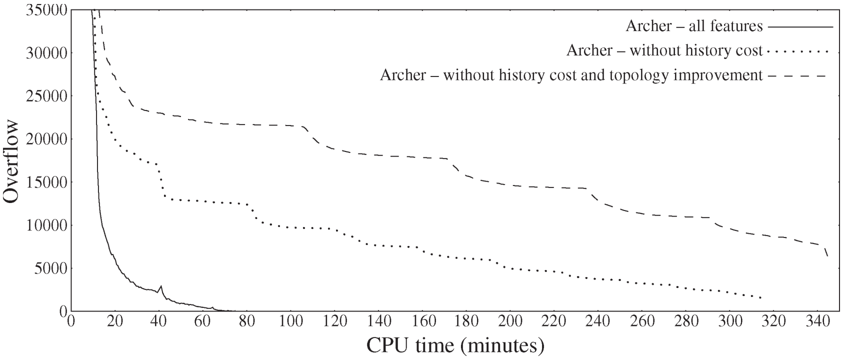


Fig. 16. Overflow values of adaptec1 during RNR iterations.

solutions, while BoxRouter 2.0 utilizes a more sophisticated ILP-based algorithm. We believe that the via counts of Archer can be reduced further by using such a sophisticated layer assignment algorithm at the end.

For both two-layer and multilayer benchmarks, Archer requires significantly less CPU times than FGR and BoxRouter 2.0. As reported in Tables IV and V, the average CPU requirements of FGR is about 4.2 times larger than Archer. Similarly, it is stated in [4] that BoxRouter 2.0 takes more than two days to complete the routing of newblue3, while it takes less than 3 h for Archer to route the same benchmark.

We have also performed an experiment on one of the benchmark circuits to demonstrate the individual impacts of the proposed features. Fig. 16 shows how the overflow values reduce during the RNR iterations. In this experiment we have used three different versions of Archer: 1) the original Archer; 2) Archer without history costs [i.e., α in (2) set to zero]; and 3) Archer without history costs and without congestion-driven topology improvement. As shown in Fig. 16, the original Archer performs nongreedy decisions at every step; however, it converges much faster than the others. In the second version, Archer tries to minimize overflows directly, instead of the history costs. However, it converges much more slowly, and it gets stuck at a local optimal at the end. It is also interesting to observe the sudden drops in the overflow values (e.g., at around time 10, 40, and 80) in this plot. These sudden drops correspond to the iterations in which we perform congestion-driven topology optimization. Finally, the third plot illustrates Archer without history costs and without topology optimization. As shown here, this version ends up with much more overflows than the others. Observe here that the overflow values drop and converge to a value periodically in this plot. The periodic drops in the overflow values correspond to the iterations in which we relax the constraints of significant number of nets, as explained in Section V-B. This experiment shows that all the new features proposed in this paper are crucial in the successful execution of Archer.

IX. Conclusion

In this paper, we propose a new global router with three main novel features. First, we propose a history-based cost metric to resolve congestion effectively. Second, we propose a framework to enable a smooth tradeoff between overflow and wirelength minimization objectives. Finally, we propose an LR-based topology improvement algorithm to minimize congestion, while maintaining a length bound.

As discussed in Section VII, our algorithms are applicable to both 2-D and 3-D grid structures. However, the most challenging publicly available benchmarks today (ISPD07 benchmarks) have some assumptions that guarantee a feasible mapping from a 2-D grid to a 3-D grid without increasing overflows [18]. Given these assumptions, the advantages of projection-based routing (Section VII-B) outweigh its disadvantages on these benchmarks. Although these assumptions may be questionable in practice, we chose to implement a projection-based approach for a fair comparison with others. For this purpose, we have used a straightforward layer assignment algorithm; however, it is possible to further reduce the number of vias for 3-D benchmarks by using a more sophisticated layer assignment algorithm, which can be a future direction for research.

References

[1] C. Albrecht, “Provably good global routing by a new approximation algorithm for multicommodity flow,” in *Proc. Int. Symp. Phys. Des.*, 2000, pp. 19–25.

[2] L. Behjat and A. Vanneli, “Steiner tree construction based on congestion for the global routing problem,” in *Proc. IEEE Int. Workshop Syst. Chip Real-Time Appl.*, 2003, pp. 28–31.

[3] C. Chiang, M. Sarrafzadeh, and C. Wong, “Global routing based on Steiner min-max trees,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 9, no. 12, pp. 1318–1325, Dec. 1990.

[4] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, “BoxRouter 2.0: Architecture and implementation of a hybrid and robust global router,” in *Proc. IEEE/ACM ICCAD*, 2007, pp. 503–508.

[5] M. Cho and D. Z. Pan, “BoxRouter: A new global router based on box expansion and progressive ILP,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 12, pp. 2130–2143, Dec. 2007.

[6] C. Chu and Y. Wong, “Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design,” in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 28–35.

[7] M. L. Fisher, “The Lagrangian relaxation method for solving integer programming problems,” *Manage. Sci.*, vol. 27, no. 1, pp. 1–18, Jan. 1981.

[8] J.-R. Gao, P.-C. Wu, and T.-C. Wang, “A new global router for modern designs,” in *Proc. ASP-DAC*, 2008, pp. 232–237.

[9] A. M. Geoffrion, “Lagrangian relaxation and its uses in integer programming,” *Math. Program.*, vol. 2, pp. 82–114, 1974.

[10] R. T. Hadsell and P. H. Madden, “Improved global routing through congestion estimation,” in *Proc. Des. Autom. Conf.*, 2003, pp. 28–31.

[11] M. Hanan, “On Steiner’s problem with rectilinear distance,” *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, 1966.

[12] M. H. Held, P. Wolfe, and H. D. Crowder, “Validation of subgradient optimization,” *Math. Program.*, vol. 6, no. 1, pp. 62–88, Dec. 1974.

[13] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, “Predictable routing,” in *Proc. Int. Conf. Comput.-Aided Des.*, 2000, pp. 110–114.

[14] S. Lee and M. D. F. Wong, "Timing-driven routing for FPGAs based on Lagrangian relaxation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 4, pp. 506–510, Apr. 2003.

[15] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *Proc. ACM Int. Symp. Field-Programmable Gate Arrays*, 1995, pp. 111–117.

[16] M. D. Moffitt, "MaizeRouter: Engineering an effective global router," in *Proc. ASP-DAC*, 2008, pp. 226–231.

[17] D. Muller, "Optimizing yield in global routing," in *Proc. Int. Conf. Comput.-Aided Des.*, 2006, pp. 480–486.

[18] G.-J. Nam, *ISPD 2007 Global Routing Contest*, 2007.

[19] M. M. Ozdal and M. D. F. Wong, "A length-matching routing algorithm for high-performance printed circuit boards," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2784–2794, Dec. 2006.

[20] M. M. Ozdal and M. D. F. Wong, "Archer: A history-driven global routing algorithm," in *Proc. IEEE/ACM ICCAD*, 2007, pp. 488–495.

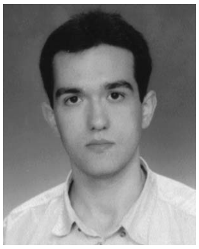
[21] M. M. Ozdal and M. D. F. Wong, "Length matching routing for high-speed printed circuit boards," in *Proc. IEEE/ACM ICCAD*, Nov. 2003, pp. 394–400.

[22] M. Pan and C. Chu, "FastRoute: A step to integrate global routing into placement," in *Proc. Int. Conf. Comput.-Aided Des.*, 2006, pp. 464–471.

[23] M. Pan and C. Chu, "FastRoute 2.0: A high-quality and efficient global router," in *Proc. ASP-DAC*, 2007, pp. 250–255.

[24] Apr. 2008. private communication with Igor Markov.

[25] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," in *Proc. IEEE/ACM ICCAD*, 2007, pp. 496–502.



Muhammet Mustafa Ozdal received the B.S. degree in electrical engineering and M.S. degree in computer engineering from Bilkent University, Ankara, Turkey, in 1999 and 2001, respectively, and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, Urbana, in 2005.

He has served in the technical program committees of International Conference on Computer Aided Design (ICCAD) and Design Automation Conference (DAC). He is currently with the Strategic CAD Labs, Intel Corporation, Hillsboro, OR. His current research interests include algorithms for computer-aided design of very large scale integration circuits, with primary focus on physical design algorithms.

Martin D. F. Wong (F'06) received the B.Sc. degree in mathematics from the University of Toronto, Toronto, ON, in 1979 and the M.S. degree in mathematics and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign (UIUC), Urbana, in 1981 and 1987, respectively.

He is currently a Professor of electrical and computer engineering with the UIUC. Before he joined UIUC, he was a Bruton Centennial Professor of computer sciences with the University of Texas at Austin. His research interests are computer-aided design (CAD) of very large scale integration (VLSI) circuits, design, and analysis of algorithms, and combinatorial optimization. He has published over 300 technical papers and has graduated 34 Ph.D. students.

Dr. Wong received the 2000 IEEE CAD Transactions Best Paper Award for his work on interconnect optimization. He also received best paper awards at the Design Automation Conference, 1986 and the International Conference on Computer Design, 1995 for his work on floorplan design and routing, respectively. His International Conference on Computer-Aided Design (ICCAD), 1994 paper on circuit partitioning has been included in the book "The Best of ICCAD—20 Years of Excellence in Computer-Aided Design" published in 2002. He has served as Technical Program Chair, General Chair, and Steering Committee member for the annual Association for Computing Machinery (ACM) International Symposium on Physical Design. He has also served on the technical program committees of numerous VLSI/CAD conferences. He has served as an Associate Editor for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN (2002–2005) and IEEE TRANSACTIONS ON COMPUTERS (1995–2000). He has also served as a Guest Editor of four special issues for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN. He is currently on the Editorial Board of ACM Transactions on Design Automation of Electronic Systems. He is an IEEE Distinguished Lecturer (2005–2006).