ARCHER: A HISTORY-DRIVEN GLOBAL ROUTING ALGORITHM

Muhammet Mustafa Ozdal

Intel Corporation
Hillsboro, OR 97124
mustafa.ozdal@intel.com

Martin D. F. Wong

Dept. of Electrical and Computer Engineering
Univ. of Illinois at Urbana-Champaign
Urbana, IL 61801
mdfwong@uiuc.edu

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 trade-off 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 show that *Archer* obtains congestion-free solutions for all circuits in the standard ISPD98 benchmarks, which is the best result published so far. Furthermore, it produces better results than the best results reported in the *ISPD-07 Global Routing Contest* in terms of routability. Compared to FastRoute [18, 19], which is the state-of-the-art RNR-based global routing algorithm, Archer improves routability by 30%, and reduces the wirelengths by 32% on the average on ISPD07 benchmarks.

1. 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 (DFM) 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.

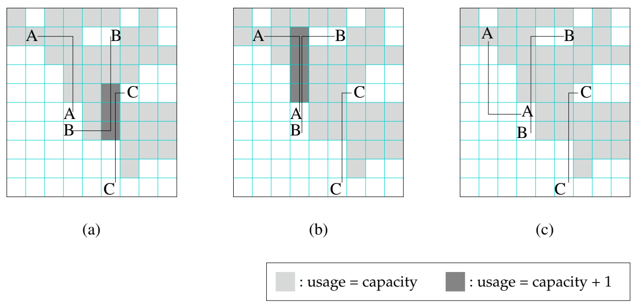

Figure 1: Routing iterations for 3 nets. (a) Initial routing with total overflow of 3. (b) Total overflow is increased to 4 after *B* is rerouted. (c) Congestion free solution is obtained after rerouting *A*.

A typical issue with the iterative algorithms is that they are likely to get stuck in local optima during the rip-up and reroute 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 overflow and wire length. Figure 1(a) illustrates an example where 3 nets are routed in 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 Figure 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. Instead, let us consider an alternative metric that will force net *B* to be rerouted as in Figure 1(b), even though it increases the total overflow. In that case, net *A* will

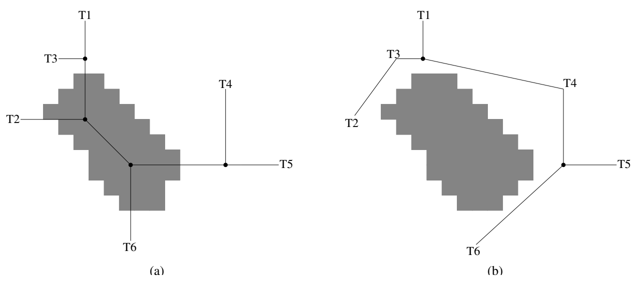

Figure 2: (a) Min-length Steiner topology for a 6-terminal net. Two of the Steiner points (shown as filled circles) are inside a congestion hotspot (shown as the shaded region). (b) An alternative Steiner topology which has larger wirelength, but avoids the hotspots.

be congested, and will be forced to be rerouted in the next iteration, as shown in Figure 1(c). Although counter-intuitive, observe here that the step that leads to the congestion-free solution is the non-greedy 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 non-greedy 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 overflow 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 Figure 1(a), net C could have been detoured to find a route with less overflow[1]. 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 trade-off 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 2. 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. Figure 2(a) illustrates 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. Figure 2(b) illustrates another topology, which is suboptimal in terms of wirelength, but avoids the heavily congested hotspot. In this paper, we propose a Lagrangian relaxation based bounded-length Steiner topology optimization algorithm to minimize overflows.

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

The rest of the paper is organized as follows. In Section 2, we present the problem formulation, and summarize the related work. Then, we outline the overview of our global routing framework in Section 3. We present our cost model in Section 4.1 to resolve congestion effectively based on congestion histories and length bounds. In Section 4.2, 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 a Lagrangian relaxation based bounded-length min-cost topology improvement algorithm to minimize overflows in Section 5. Finally, we present our experimental results in Section 6.

2. PROBLEM FORMULATION AND RELATED WORK

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 two dimensional grid model is used, then the routing tracks on every layer are lumped together to compute the edge capacities. On the other hand, a three dimensional 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 three dimensional grid model can capture the capacity differences of different layers, it requires layer assignment to be performed during global routing. The algorithms we propose in this paper are applicable to both two and three dimensional models.

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, 14]. 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 [18, 19] is a recently proposed RNR-based iterative algorithm, and the results reported in [18, 19] outperform other existing iterative algorithms such as Labyrinth [11] and Chi Dispersion router [8]. However, it suffers from the common problems outlined in Section 1. Specifically, the order in which nets are routed can have significant impact on the routing qualities. Also, the tradeoff between the objectives of overflow and wirelength minimization is not focused on. In our experiments, we demonstrate average of 30% overflow improvements, and 32% wirelength improvements compared to FastRoute on ISPD07 routing benchmarks.

BoxRouter [4] is also a recently proposed global router that outperforms existing 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 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 2 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. However, 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 Figure 2, the Steiner trees that are suboptimal in terms of wirelength can in fact lead to lower congestion values.

FastRoute [18] also utilizes some techniques to generate congestion-driven Steiner trees. The min-length Steiner tree generation algorithm FLUTE [5] 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 Figure 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 illustrated in the example of Figure 1. The idea of using congestion histories has been used in different contexts such as FPGA routing [12, 13] and package routing [16, 17]. 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.

```
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 [5]
   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 $n$ to minimize congestion (Sec. 5)
   for each congested 2-pin connection $c$ do
      ripup $c$
      reroute $c$ to minimize Equation 1 (Sec. 4.1)
      based on congestion history of $c$:
         update constraints of $c$ as in Figure 5 (Sec. 4.2)
```

Figure 3: The high-level global routing framework

3. OVERVIEW OF METHODOLOGY

The high level framework for the global router we propose in this paper is outlined in Figure 3. In the beginning, we use FLUTE [5] to generate a min-length Steiner tree for each net, and we route each topology edge[2] with the objective of minimizing wire lengths. Then, we perform rip-up and reroute (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 ripup each congested 2-pin connection, and we try to reroute it to minimize the cost function defined in Equation 1, as will be explained in Section 4.2. 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 (Figure 6), 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 (Figure 7) or maze routing, while increasing the length bounds gradually. This framework is outlined in Figure 5, which is explained in detail in Section 4.2.

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 [18]. Note here that topology generation for a net is a more expensive operation than routing 2-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.

4. GLOBAL ROUTING BASED ON CONGESTION HISTORIES

4.1. 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 total overflow and wirelength. However, as mentioned earlier, increasing wirelengths

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

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 4.2. At any point during rip-up and reroute (RNR) iterations, we use the following cost metric to route a 2-pin connection, while making sure that the respective length bound is satisfied:

1. Minimize the total congestion history cost of the utilized edges.
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, one can think that this approach 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 [13], congestion history cost of grid edge e in iteration k is defined as follows:

$$cost(e) = (1 + \alpha . h_e^k) \times overflow(e) \qquad (1)$$

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:

$$h_e^k = \begin{cases} h_e^{k-1} & \text{if edge } e \text{ is congestion free in iteration } k \\ h_e^{k-1} + k & \text{if edge } e \text{ has nonzero overflow in iteration } k \end{cases} \qquad (2)$$

So, 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 Equation 1, 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 non-greedy decisions for the purpose of getting closer to the global optimal.

Let us go back to the example of Figure 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. So, it will be rerouted as in part (b), even though the total overflow increases from 4 to 5. 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 Equation 1 as a function of iteration index, as illustrated in Figure 4. The value of α determines the type of optimization being

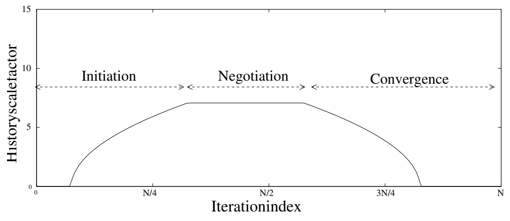

Figure 4: The plot of history scale factor α as a function of iteration index. A small α value shifts the focus to overflow minimization objective, while a large value increases the importance of congestion history costs.

done in the low level routing iterations. Based on the value of α, we can categorize our iterative algorithm into 3 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 Equation 1, 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 Equation 1), 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 Figure 4.

4.2. History-Based Routing of 2-Pin Connections

After generating the Steiner topologies for each net, we iteratively rip-up and reroute the 2-pin connections of the nets using the cost function described in Section 4.1. Here, there are two important considerations that need to be taken into account:

- Pattern routing is significantly faster than maze routing. For example, it is reported [18] 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.
- Increasing wirelengths prematurely to avoid congestion can eventually lead to higher overflows. For example, consider net C in Figure 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 Figure 1(c). So, 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 2-pin connections, depending on their congestion histories, as outlined in Figure 5. Here, each

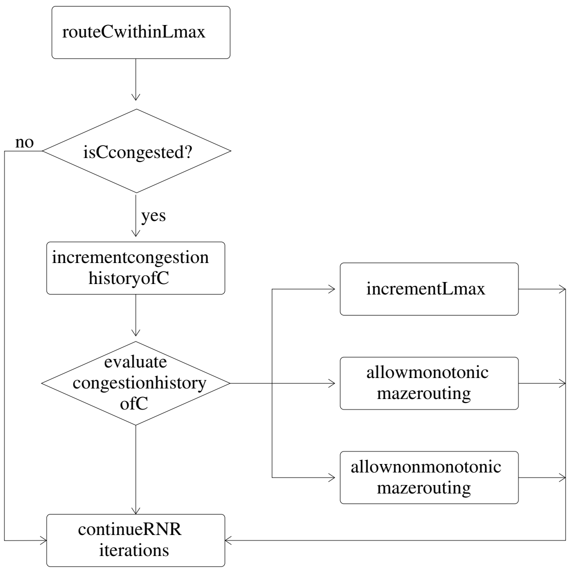
routeCwithinLmax
isCcongested?
no
yes
incrementcongestion historyofC
evaluate congestionhistory ofC
incrementLmax
allowmonotonic mazerouting
allownonmonotonic mazerouting
continueRNR iterations

Figure 5: Routing methodology for a 2-pin connection. Initially, L_{max} is set to the Manhattan length of the connection, and only pattern routing is allowed.

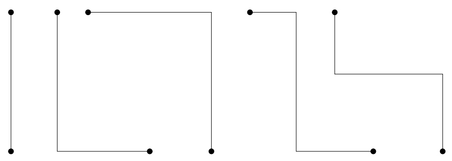

Figure 6: Min-length pattern routing (I, L, Z routing).

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.

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 illustrated in Figure 6. 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 illustrated in Figure 7. 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 illustrated in Figure 8(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.

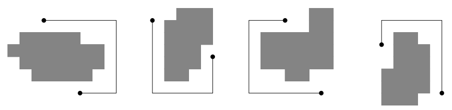

Figure 7: Pattern routing with detours (U routing).

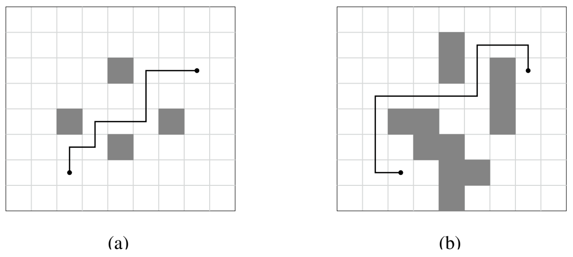

Figure 8: (a) Monotonic maze routing. (b) Non-monotonic maze routing.

Non-monotonic maze routing: If a connection is still congested even after several iterations of monotonic maze routing, we gradually increase its length bound to perform non-monotonic maze routing. Depending on the length bound of the connection, we define a bounding box around the terminals, and perform non-monotonic maze routing within this box, as illustrated in Figure 8(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.

5. CONGESTION DRIVEN TOPOLOGY OPTIMIZATION

In this section, we propose a Lagrangian relaxation based bounded-length minimum-cost topology improvement algorithm, which is outlined in Figure 9. Here, the objective is to create a topology for net n that minimizes the objective function defined in Equation 1 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 4.2.

The algorithm outlined in Figure 9 starts with creation of the Hanan grid [9] based on the terminal positions of net n. After the Hanan grid is created, congestion costs are assigned to each edge based on Equation 1. 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 rip-up and reroute (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 over-constraint. 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 a Lagrangian relaxation based iterative improvement algorithm.

Lagrangian relaxation (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

```
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 4
    update $\lambda$ based on length of $T$
  return best topology $T_{best}$ that satisfies $L_{max}$ bound
```

Figure 9: LR-based bounded-length min-cost topology improvement algorithm

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. So, 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 [6, 7].

The main objective here is to find the best Steiner topology that minimizes the sum of congestion history costs, as defined in Equation 1, under the length constraint of L_{max}:

$$\begin{aligned} &minimize && \sum_{e \in \mathcal{T}} cost(e) \\ &subject\ \ to: && \\ & && \sum_{e \in \mathcal{T}} length(e) \leq L_{max} \end{aligned} \tag{3}$$

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 Equation 1. If we apply LR on this formulation, our objective becomes the minimization of:

$$\sum_{e \in \mathcal{T}} cost(e) + \sum_{e \in \mathcal{T}} \lambda.length(e) \tag{4}$$

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:

$$\lambda^{k+1} = max(0, \lambda^k + t_k \times \begin{cases} 1 & \text{if} \quad length(T) > L_{max} \\ -1 & \text{if} \quad length(T) \leq L_{max} \end{cases} \tag{5}$$

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 [10],

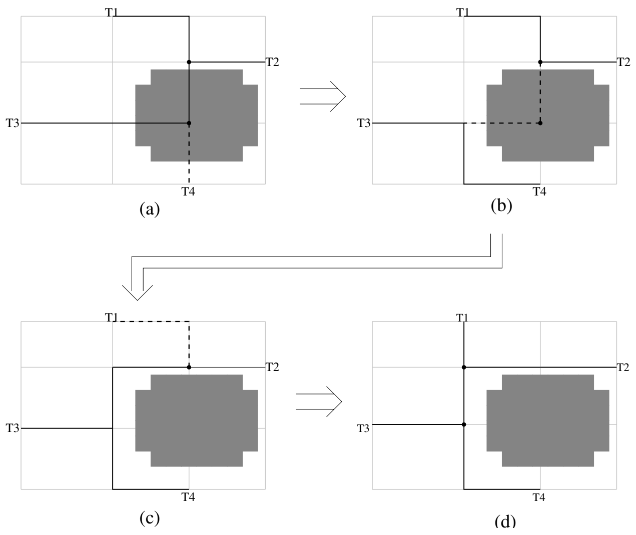

Figure 10: An example execution of the topology improvement algorithm. The dashed lines represent the connections to be ripped up in the next step. The shaded region represents the congested hotspot.

which states that as $k \rightarrow \infty$, it should be the case that $t_k \rightarrow 0$ and $\sum_{i=1}^{k} t_i \rightarrow \infty$. In particular, we have used $t_k = 1/k^{\alpha}$ in our experiments, where $\alpha < 1$ is a constant to provide a trade-off between solution quality and convergence speed. In our experiments, we have set α to 0.1.

Based on the objective function given in Equation 4, we iteratively rip-up and reroute the connections of T to minimize the LR cost. To choose the final topology, we utilize a hill-climbing technique. As mentioned before, non-greedy steps are important to avoid getting stuck at local optima. The LR cost function enables these non-greedy 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.

Figure 10 illustrates 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 3 connections that need to be routed through the congested region. First, a non-greedy 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.

6. EXPERIMENTAL RESULTS

We have implemented our algorithms in C++, and performed experiments on a computer with Intel Xeon 3.60Ghz processor and Linux operating system.

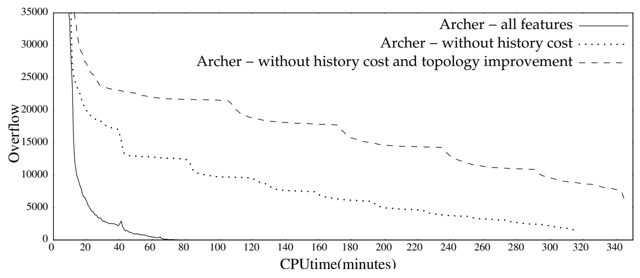

Figure 11: Overflow values of adaptec1 during RNR iterations.

We have performed 2 sets of experiments. First, we have run our global router *Archer* on ISPD98 benchmarks, and compared our results with the state-of-the-art global routers, as shown in Table 2. Observe that Archer successfully resolves the congestion in all circuits, while FastRoute [19] and BoxRouter [4] fail to find congestion-free routing solutions in several of the circuits. Furthermore, Archer outperforms state-of-the-art routers not only in terms of total overflows, but also in terms of total wirelength, as shown in this table. Note that the set of parameters used in Archer are identical for all circuits in this table.

Our second set of experiments have been performed on ISPD07 benchmarks, which were released only recently, as part of *ISPD-07 Global Routing Contest* [15]. Each circuit in these benchmarks has a 2-layer version, and a multi-layer version. Here, all layers have identical width and spacing constraints, and 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 multi-layer circuit. Hence, it is possible to project the solution of a 2-layer circuit to the corresponding multi-layer circuit through layer assignment. This projection can be performed in a straightforward way without increasing the overflow values and the total wirelengths. Due to page limitations, the details of this operation will not be given. Table 1 displays our results on these benchmark circuits. Observe that the overflow values and the wirelengths are identical for the 2-layer and multi-layer versions of the same circuit; however, there are more vias in the multi-layer solutions. Note that the set of parameters used in Archer are identical for all circuits in this table, except for *newblue3*, which is assigned an earlier termination condition due to the fact that it is unroutable.

We also compare our results with the best results reported in the *ISPD-07 Global Routing Contest* in Table 3. 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 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 reported as the actual wirelength plus 3 times the number of vias. For one-to-one comparison with other routers, the wirelengths reported in Table 3 are computed using this metric.

FGR and *MaizeRouter* are the winners of the ISPD-07 global routing contest, and both algorithms have not been published yet. The results reported for *BoxRouter* are from an extension of the original algorithm [4], with major enhancements that are not published yet. Similarly, *FastRoute* is an extension of [18, 19], with possible unpublished enhancements. Note here that FastRoute is the best state-of-the-art global routing algorithm in the literature that is based on rip-up and reroute (RNR). The results reported in [18] are significantly better than other similar RNR-based algorithms such as [8, 11].

Table 1: Archer results on ISPD07 benchmarks, with the same set of parameters used for all circuits.

	Common Results		2-Layer Circuits		Multi-layer Circuits	
Name	ovfl	length	# via	cpu (m)	# via	cpu (m)
adaptec1	0	3736K	697K	86	2548K	87
adaptec2	0	3354K	702K	22	2634K	23
adaptec3	0	9702K	1280K	49	4902K	51
adaptec4	0	8921K	1235K	11	4412K	12
adaptec5	0	10612K	1879K	246	7599K	248
newblue1	682	2452K	796K	49	3052K	50
newblue2	0	4629K	1054K	6	4007K	7
newblue3	33394	7565K	1120K	162	4104K	163

As shown in Table 3, Archer obtains the lowest overflow values compared to other global routers[3]. The wirelength results of Archer, FGR, MaizeRouter, and BoxRouter are comparable to each other, and they are much lower than those of FastRoute. Unfortunately, further detailed comparison between these routers is not possible, since the algorithms of FGR and MaizeRouter, and the new features of BoxRouter have not been published yet. However, it is interesting to observe the difference between the results of Archer and FastRoute, which are both based on RNR. Archer results in 30% improvements in terms of overflow, and 32% improvements in terms of wirelength, compared to FastRoute.

We have also performed an experiment on one of the benchmark circuits to demonstrate the individual impacts of the proposed features. Figure 11 illustrates how the overflow values reduce during the RNR iterations. In this experiment we have used 3 different versions of Archer: 1) the original Archer, 2) Archer without history costs (i.e., α in Equation 1 set to 0), and 3) Archer without history costs and without congestion-driven topology improvement. As shown in Figure 11, the original Archer performs non-greedy 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 4.2. This experiment shows that all the new features proposed in this paper are crucial in the successful execution of Archer.

7. CONCLUSIONS

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 ob-

[3]Since it is possible to map the solutions of 2-layer circuits to the corresponding multi-layer circuits, only 2-layer results are shown in Table 3 due to page limitations. Our results for multi-layer circuits are given in Table 1.

Table 2: Experimental results on ISPD98 benchmarks

			Archer			FastRoute			BoxRouter			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 3: Comparison with the results of ISPD-07 Global Routing Contest

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

jectives. Finally, we propose a Lagrangian relaxation based topology improvement algorithm to minimize congestion, while maintaining a length bound. Our experiments on public benchmarks show significant improvements compared to the state-of-the-art algorithms in the literature.

8. REFERENCES

[1] C. Albrecht. Provably good global routing by a new approximation algorithm for multicommodity flow. In *Int'l Symposium on Physical Design*, pages 19–25, 2000.

[2] L. Behjat and A. Vanneli. Steiner tree construction based on congestion for the global routing problem. In *Proc of IEEE Int'l Workshop on System on Chip for Real-Time Applications*, pages 28–31, 2003.

[3] C. Chiang, M. Sarrafzadeh, and C. Wong. Global routing based on steiner min-max trees. *IEEE Transactions on Computer Aided Design*, 9(12):1318–1325, 1990.

[4] M. Cho and D. Z. Pan. Boxrouter: A new global router based on box expansion and progressive ilp. In *Proc. of Design Automation Conference*, pages 373–378, 2006.

[5] C. Chu and Y. Wong. Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design. In *Proc of Int'l Symp. on Physical Design*, pages 28–35, 2005.

[6] M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[7] A. M. Geoffrion. Lagrangian relaxation and its uses in integer programming. *Mathematical Programming*, 2:82–114, 1974.

[8] R. T. Hadsell and P. H. Madden. Improved global routing through congestion estimation. In *Proc. of Design Automation Conference*, pages 28–31, 2003.

[9] M. Hanan. On steiner's problem with rectilinear distance. *SIAM Journal of Applied Mathematics*, 14:255–265, 1966.

[10] M. H. Held, P. Wolfe, and H. D. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.

[11] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. Predictable routing. In *Proc. of Int'l Conf. on Computer Aided Design*, pages 110–114, 2000.

[12] S. Lee and M. D. F. Wong. Timing-driven routing for FPGAs based on lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 506–510, 2003.

[13] L. McMurchie and C. Ebeling. Pathfinder: A negotiation-based performance-driven router for FPGAs. In *Proc. of ACM Int'l Symp. on Field-Programmable Gate Arrays*, pages 111–117, 1995.

[14] D. Muller. Optimizing yield in global routing. In *Proc. of Int'l Conf. On Computer Aided Design*, pages 480–486, 2006.

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

[16] M. M. Ozdal and M. D. F. Wong. A length-matching routing algorithm for high-performance printed circuit boards. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 25:2784–2794, 2006.

[17] M. M. Ozdal and M. D. F. Wong. Length matching routing for high-speed printed circuit boards. In *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 394–400, Nov. 2003.

[18] M. Pan and C. Chu. Fastroute: A step to integrate global routing into placement. In *Proc. of Int'l Conf. on Computer Aided Design*, pages 464–471, 2006.

[19] M. Pan and C. Chu. Fastroute 2.0: A high-quality and efficient global router. In *Proc. of ASP-DAC*, pages 250–255, 2007.