MAIZEROUTER: Engineering an Effective Global Router

Michael D. Moffitt

***Abstract*—In this paper, we present the complete design and architectural details of MAIZEROUTER. MAIZEROUTER reflects a significant leap in progress over existing publicly available routing tools yet relies upon relatively simple operations, including *extreme edge shifting*, a technique aimed primarily at the efficient reduction of routing congestion, and *edge retraction*, a counterpart to extreme edge shifting that serves to reduce unnecessary wirelength. We present enhanced variations of these operations to enable the *rapid exploration* of candidate paths, along with a form of *dynamic cost deflation* that provides our various path computation procedures with progressively more accurate (and less optimistic) cost information as search continues. These algorithmic contributions are built upon a framework of *interdependent* net decomposition, a representation that improves upon traditional two-pin net decomposition by preventing duplication of routing resources while enabling cheap and incremental topological reconstruction. Collectively, these operations permit a broad search space that previous algorithms have been unable to achieve, resulting in solutions of considerably higher quality than those of well-established routers.**

***Index Terms*—Algorithms, design automation, optimization, routing.**

I. INTRODUCTION

GLOBAL routing is a critical step in modern very large scale integration physical design. Its importance has recently been cast into the limelight with the IEEE Council on Electronic Design Automation-sponsored Global Routing Contest [30], a competition that attracted nearly a dozen academic and industrial participants from around the globe. As reported in EE Times [9], contests such as these serve as an important bellwether of urgent problems in Electronic Design Automation, as well as a showcase for state-of-the-art algorithms and solutions.

Despite increased attention to the problem of global routing, there remains little consensus as to what techniques contribute to a truly successful routing engine. This can be seen not only in the wide range in solution quality of entries to the competition but also in the broad spectrum of algorithms that have been proposed in recent years. A survey of conventional approaches to global routing reveals a variety of methods, ranging from traditional maze-based algorithms [12] to flow-based techniques [1], to integer linear programming (ILP) formulations [5], and to congestion-driven Steiner tree generation [22]. The choice of algorithm has a dramatic effect on competing qualities of the solution (i.e., overflow and wirelength), as well as the runtime incurred by the solver.

Of equal importance to the design of an effective global router are the various data structures and elementary atomic units used "under the hood" to model and maintain a partial (or complete) solution. For instance, several academic routers require a strict decomposition of Steiner trees into two-pin nets, while others instead operate directly on an explicitly defined topology. Surprisingly, little attention has been paid in the literature to the mechanisms needed to manipulate these representations dynamically during the course of a global routing algorithm. This is due, in part, to limitations imposed by previous global routers on the flavor of manipulations allowed. In particular, topological reconstruction is usually avoided at all costs, a design decision that simplifies the construction of algorithms but severely curtails the freedom of the routing engine.

In this paper, we reveal the complete design and architectural details of MAIZEROUTER [20], a novel state-of-the-art multilayer global routing algorithm that took First Place in the 3-D track of the inaugural Global Routing Contest held at the International Symposium on Physical Design (ISPD) 2007. At the highest level, the design of MAIZEROUTER draws primarily upon two complementary edge-based operations:

- **extreme edge shifting**: a simplification and generalization of edge shifting [22] that has been enhanced to restructure Steiner tree topologies, providing particularly effective support for congestion reduction;
- **edge retraction**: a counterpart to extreme edge shifting that reduces unnecessary wirelength by safely sliding tree segments into areas where overflow has been eliminated.

We generalize these operations to enable the **rapid exploration** of candidate paths, and expose parallels between our methods and the practice of pattern routing. We also propose an application of dynamic congestion amplification [10] to the powerful logistic function used in [22], serving to balance the tradeoff between wirelength and overflow as a function of runtime.

These algorithmic contributions are situated atop a framework of **interdependent net decomposition**, a model that improves upon traditional two-pin net decomposition by preventing duplication of routing resources while enabling cheap and incremental topological reconstruction. We introduce two

Manuscript received October 1, 2007; revised January 31, 2008, April 12, 2008, and June 6, 2008. Current version published October 22, 2008. This paper was recommended by Associate Editor L. Scheffer.

The author is with the Design Productivity Group, IBM Austin Research Laboratory, Austin, TX 78758 USA (e-mail: mdmoffitt@us.ibm.com).

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.2008.2006082

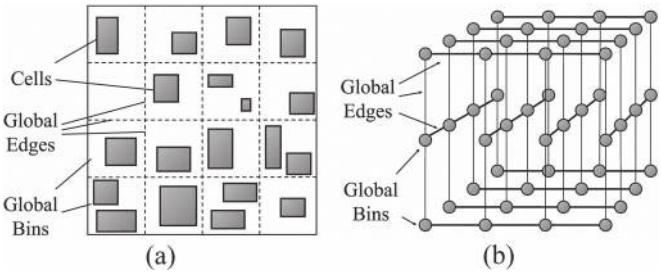

Fig. 1. Bin decomposition and grid graph of the global routing problem formulation. (a) Global bin decomposition. (b) Corresponding grid graph.

mechanisms needed to adequately maintain this internal representation:

- **garbage collection**: a process whereby leftover routing segments produced by our edge-based operations are removed from the solution;
- **net defragmentation**: a means to consolidate adjacent routing segments for the purpose of wirelength recovery.

Collectively, these operations permit a broad search space that previous algorithms have been unable to achieve. Combined with a moderate amount of traditional maze routing, our algorithm is shown to surpass previous routers in solution quality and remains extremely competitive in runtime. On the ISPD '98 benchmarks, MAIZEROUTER achieves zero overflow (i.e., full routability) on all ten instances, running an average of 7.8× faster than *BoxRouter* and producing 1.84% shorter wirelength. In addition, a reduction in wirelength of 2.73% is observed over *FastRoute* 2.0. Competitive results on the ISPD '07 benchmarks are observed as well.

The remainder of this paper is organized as follows. Section II covers preliminary background on global routing, including a basic problem formulation, previous algorithms, and leading solvers. Section III introduces our algorithm, namely, MAIZEROUTER, and its principle algorithmic components. In Section IV, we expose the underlying representation that MAIZEROUTER uses to encode and manipulate its routing solutions, along with the basic procedures for maintaining this structure. In Section V, we present an empirical comparison of algorithms. We also provide a complete and thorough summary of the Global Routing Contest, including performance statistics of all entries to the competition. We briefly describe future work in Section VI and end with concluding thoughts in Section VII.

II. BACKGROUND

A. Global Routing: Problem Formulation

The problem of global routing can be characterized as follows. There is a grid graph G specifying a set of vertices V and a set of edges E. As shown in Fig. 1, each vertex $v_i \in V$ corresponds to a particular rectangular 3-D region (or cell) of the global routing resource grid, and each edge $e_{ij} \in E$ corresponds to a boundary between adjacent vertices (with a maximum allowable resource m_{ij}). There is also a set of nets N, where each net $n_i \in N$ is composed of a set P_i of pins (with each pin corresponding to a vertex v_i). A *solution* is a mapping of nets to routes, in which each route connects all the pins of a net using the edges of the graph G.

When evaluating a routing solution (or, for that matter, a routing engine), one is typically concerned with three metrics. *Overflow* refers to the total amount of demand that exceeds capacity over all edges [16]. As it directly corresponds to the routability of the design, overflow is desired to be as small as possible (ideally zero). *Wirelength* is the combined length of segments needed to route all nets and should also be minimized. In 3-D routing, this calculation can also include special costs for *vias*, the wires used to connect routing segments between consecutive layers of metal. Finally, one is almost always concerned with the *runtime* needed to construct the solution. This is particularly true in cases where global routing is repeatedly used to guide a placement algorithm [24]. Global routing is a textbook example of a multiobjective optimization problem, in which the relative importance of the individual criteria depends heavily on the context in question.

B. Global Routing: Basic Algorithms

As is the case with many large-scale optimization problems, a wide variety of algorithms have been proposed for global routing [12], [26]. Here, we briefly review the most well-known and successful techniques and solvers, focusing particularly on those methods that directly relate to our proposed contributions.

Maze routing is a grid-based search algorithm that has long held a reputation as a brute-force approach to routing, connecting pairs of source and target locations using the shortest possible path (with respect to an arbitrary cost function). Naïve implementations typically employ Breadth First Search (BFS) or Dijkstra's algorithm, whereas A*-style approaches (using, for instance, the Manhattan distance between two points as an admissible heuristic) often perform significantly better.

Pattern routing [16] considers a significantly smaller number of paths than does maze routing, in an attempt to increase the speed of the routing algorithm. Simple patterns (such as one-bend L shapes and two-bend Z shapes) allow substantially fewer grid edges to be examined. However, pattern routing offers no guarantee on the optimality of the chosen path and must typically be used in conjunction with some amount of maze routing to generate solutions of adequate quality.

After the initial routes for a set of nets have been determined, they may be repeatedly torn apart and reassigned in an iterative repair framework known as *Ripup-and-Reroute* (R&R). R&R strategies often differ by the order in which to visit nets, as this ordering may significantly impact the allocation of resources.

Among the more exotic approaches to global routing is its reformulation as a *multicommodity flow* problem [1]. Here, the flow problem is used to solve a linear programming relaxation of global routing, whose dual solution provides a lower bound on the optimum maximum relative congestion.

Due to the computational expense of global routing, some have explored the use of *probability-based congestion prediction* [14], [19], [27], [28] in an effort to help placement algorithms anticipate which regions of the chip will present the most difficult areas for routability. Although recent work has cast doubt on the usefulness of this technique [29], it is still commonly used in industrial placement tools.

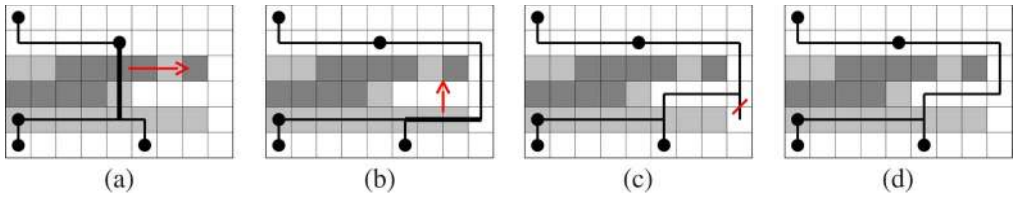

Fig. 2. Two applications of *extreme edge shifting* followed by *garbage collection*. Dark gray indicates an area of overflow, light gray indicates an at- or near-capacity area, and white indicates an area of relatively low demand.

C. Global Routing: Leading Solvers

Prior to 2006, the leading global routing tools were Labyrinth [15] (which uses primarily pattern routing) and the *Chi* Dispersion Router [10] (which incorporates a form of *congestion amplification* into its flow). However, significant progress has been made in recent years resulting in a new breed of state-of-the-art routers.

BoxRouter [5] progressively expands a box initiated from the most congested region of the chip, applying an ILP formulation to reroute wires between successive boxes. Although the ILP considers only L-shaped patterns for each two-pin decomposition, a round of maze routing is applied thereafter to compute paths for wires that cannot be successfully routed.

FastRoute [22] uses a congestion map to warp the structure of a Hanan grid [11] during Steiner tree generation, followed by edge shifting and a form of pattern routing. It has also recently been enhanced with monotonic routing and multisource multi-destination maze routing [23], although the most recent version of *FastRoute* did not prove to be competitive in the contest.

Along with our own MAIZEROUTER, several other modern routers debuted at the Global Routing Contest, including a sequel to *BoxRouter* [4] and a new router named FGR [25]. Two others—*Archer* [21] and *NTHU-Route* [8]—emerged afterward, both based on R&R and history-based congestion functions. The latter also employs adaptive multisource, multi-destination maze routing, and congestion-based net reordering. Recently, released results demonstrate substantially improved quality as compared with the contest solutions, reflecting state-of-the-art performance.

With the exception of FGR, all modern routers make use of the publicly available *FLUTE* package [6], [7] to create initial Steiner trees. *FLUTE* uses lookup tables to produce solutions of optimal wirelength for nets containing up to nine pins and applies a divide-and-conquer strategy for nets of larger size.

III. MAIZEROUTER—ALGORITHM FUNDAMENTALS

While recent work has certainly broadened the space of high-level techniques available to global routers, several deficiencies remain. In this section, we describe the algorithmic details and overall flow of our routing engine, named MAIZEROUTER, which addresses many of these issues.

A. Solution Initialization and Representation

MAIZEROUTER begins by greedily generating complete fully connected routes for all nets independently from one another. In our implementation, we use *FLUTE* to derive the topology for each net, although any reasonably efficient Rectilinear Steiner minimal tree (RSMT) or Rectilinear Minimum Spanning Tree package (such as FastSteiner [13]) will suffice. Since virtually no attempt is made to improve routability at this stage, the cheap initial solution typically comes at the expense of an extremely high amount of overflow.

Conventional wisdom stipulates that during the entire course of a routing procedure, the topology of any individual net may be understood and represented recursively as a tree. As such, we will likewise refer to Steiner points, tree edges, etc., to describe the remainder of the algorithm. However, in understanding the search space of MAIZEROUTER, it is more useful to imagine the routing of a net simply as an unnested collection of intervals (or flat wires) in 2-D (or, for multilayer routing, 3-D) space. As will be addressed in Section IV, we do not explicitly encode net topology, as we instead operate on flat segments obtained from a special type of decomposition. Hence, our search paradigm will be one where these flat wires are individually rerouted in such a way that reduces (or eliminates) overflow while preserving connectivity of their corresponding nets.

B. Extreme Edge Shifting

Of the many techniques that the engine *FastRoute* [22] employs, one particularly useful step called *edge shifting* is designed to move tree edges out of highly congested regions.[1] The approach leverages the Steiner tree topology in such a way that guarantees no change in wirelength. For instance, consider the Steiner tree in Fig. 2(a), which happens to lie in an area of high congestion. Edge shifting will permit the bold edge to be slid anywhere between its current position and the far left side of the diagram, since this area is bounded from above and below by sibling segments. If the cumulative cost of any of these alternate positions is more desirable than the current location, the edge may be safely relocated. Hence, edge shifting provides a means to reroute the path between a pair of points by exploiting the presence of neighboring wires.

Although powerful, edge shifting is sharply limited in scope, in that it provides a relatively narrowband where the tree edge may be repositioned. In our current example, there is no place within the so-called "safe region" where overflow can be completely avoided. As a result, the effectiveness of the router may remained burdened by heavy congestion in this area.

In response, we introduce a critical generalization of edge shifting that we call *extreme edge shifting*. Extreme edge shifting relaxes the requirement that Steiner wirelength be preserved when moving an existing segment out of a congested region.

[1] A similar technique named *segment move* is deployed in the DpRouter engine [2].

In fact, the new edge may be relocated far outside the original tree, so long as the appropriate routing segments are added to connect it to the points of origin (forming a C-shaped detour).[2] As illustration, Fig. 2(b) shows a case where the bold edge from Fig. 2(a) has been moved to the far right, a region where routing resource is relatively underutilized. Two so-called *parallel segments* must be added to join the *central segment* to the tree. A second application of extreme edge shifting on another segment of the tree is shown in Fig. 2(c), this time in an upward direction.

One may choose from any number of strategies for cost() (i.e., step, linear, etc.) although our implementation makes use of the logistic function. Just as in traditional edge shifting, cost need not be accumulated for any cell that contains a wire for the current net. A single pass of extreme edge shifting will examine each cell in the grid, and if the ratio of demand to capacity is above a particular threshold, it will attempt to detour as many segments away from that cell as possible. As described earlier, the algorithm focuses only on individual segments that pass through the region of congestion and will not explicitly attempt to manipulate or reroute the entire tree of any net (as would typically be done in R&R). We perform several such passes of extreme edge shifting to achieve its full benefit.

C. Edge Retraction

There are two notable disadvantages of our repair procedure that require remedy. The first of these is the creation of superfluous wires, such as the dangling segment shown in Fig. 2(c). We will address this concern in Section IV-B when presenting the underlying representation used by MAIZEROUTER to incrementally maintain routing solutions. The second major deficiency of extreme edge shifting is that, depending on how the cost function has been adjusted to balance overflow and wirelength, it may produce very long parallel wires. Of course, this happens for good reason, namely, to route the central segment toward a distant region that is less congested, thereby reducing overflow and freeing resources for other wires. However, as the engine begins to reach a routable (or possibly near-routable) solution, it becomes more important to recover whatever wirelength has been sacrificed in intermediate steps.

Our solution is to reverse the process of extreme edge shifting, in an attempt to "undo" its adverse effects. This new procedure, deemed *edge retraction*, is identical to extreme edge shifting with two exceptions. First, the segment being moved must remain bounded from above and below by neighboring wires.[3] Second, we will move this segment only so far as it will not create overflow in any of the cells in its new position, thus maintaining whatever degree of routability had been previously achieved. Once the segment has been assigned its new position, unneeded wires may be removed from the net. Edge retraction is similar in spirit to the postrouting stage of *BoxRouter* [5], since both reduce wirelength as a postprocessing step (although our approach avoids the use of maze routing).

[2]A form of U-shaped move is proposed in [18], but it too is constrained by the predetermined topology of the net.

[3]Since the objective of edge retraction is to reduce wirelength, we will not permit a location that necessitates the addition of segments.

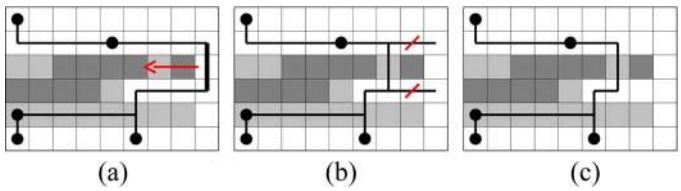

Fig. 3. Example of *edge retraction* followed by *garbage collection*. Dark gray indicates an area of overflow, light gray indicates an at- or near-capacity area, and white indicates an area of relatively low demand.

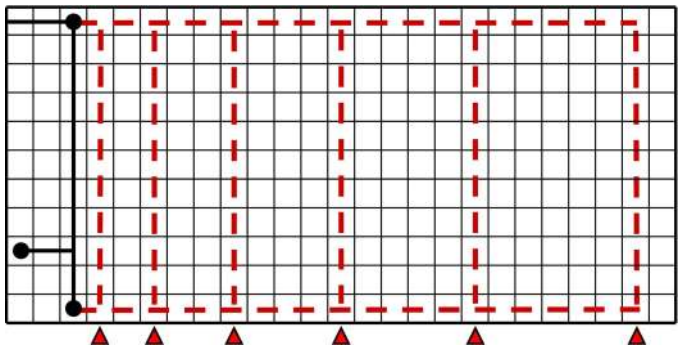

Fig. 4. Space of candidate paths considered by rapid exploration when the partial set of possible central segment locations $C = \{1, 3, 6, 10, 15\}$.

In Fig. 3, we demonstrate edge retraction on our running example. The segment on the far right can be moved two units to the left without incurring additional overflow [Fig. 3(a)]. After this translation is performed, dead-end segments are detected and eliminated [Fig. 3(b)], thereby fulfilling the promise of reduced wirelength. The final routing for this net is shown in Fig. 3(c).

D. Relations to Pattern Routing: Rapid Exploration

As discussed earlier, both the classical and extreme versions of edge shifting used in *FastRoute* and MAIZEROUTER, respectively, leverage the existence of nearby subnets when repositioning routing segments. However, there is an additional computational advantage to extreme edge shifting (as compared with, for instance, maze routing) that resembles the well-known practice of *pattern routing* [16], which considers L- and Z-shaped solutions to two-pin nets. Recall that while maze routing requires $O(NlgN)$ time[4] in the number of cells N, pattern routing requires only $O(N)$ time and will typically examine far fewer edges. For instance, the cost of L-shaped routes can be computed by examining only the edges along the periphery of the bounding box. Similarly, Z-shaped routes examine only half of the interior edges (i.e., those in alignment with the center of the Z). The C-shaped routes explored by MAIZEROUTER provide a similar computational advantage, ignoring all edges that do not fall along either the parallel segments or each candidate central segment.

It is with this observation in mind that we propose a slightly more generalized version of edge shifting that enables the *rapid exploration* of candidate paths. While accumulating the cost of the parallel segments, we compute only a partial set of possible

[4]This is for arbitrary cost functions; pure wirelength minimization can be performed with a linear-time BFS.

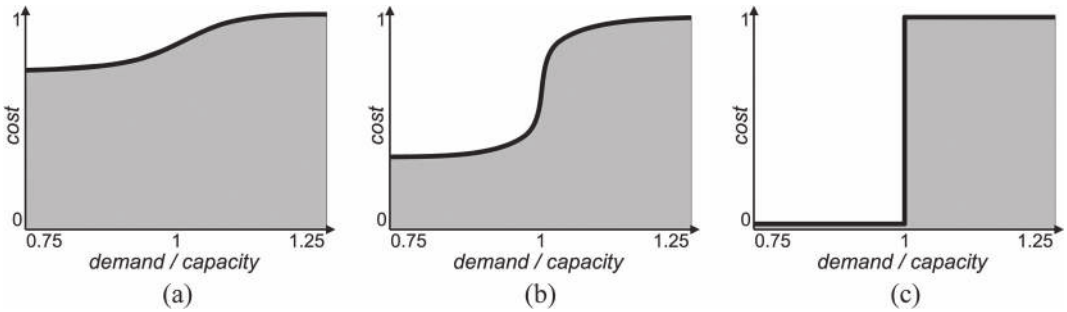

Fig. 5. Dynamic logistic cost function of MAIZEROUTER. (a) To distribute routing resources and maintain reasonable wirelength, low values of h and k are used to inflate congestion information in early stages of search. (b) As search continues, h and k are steadily increased to encourage fewer violations. (c) In the final stages of search, high values of h and k approximate the step function, corresponding to an accurate evaluation of overflow.

locations for the central segment and thus examine an even fewer number of global edges. As shown in Fig. 4, this allows one to sample the search space the more sparsely as longer detours are attempted.

E. Dynamic Cost Deflation

When accumulating total cost along a path, our edge manipulation algorithms employ a variation on the logistic function (as suggested in [22]) to amplify congestion estimates

$$\text{cost} = 1 + \frac{h}{1 + e^{-k(\text{demand}-\text{capacity})}} - h.$$

Here, parameter k controls the smoothness of the function, with larger values causing it to change more rapidly in the center. Parameter h controls the tradeoff between congestion and wirelength; a value of zero results in path measurements based solely on total length, whereas a value of one measures solutions entirely by total congestion. The use of functions to amplify congestion information, originally proposed in [10], helps immensely in the early stages of search, as it provides the router with an artificial incentive to focus on trouble areas.

However, it is also observed in [10] that the benefits of congestion amplification diminish in later iterations, when the global router should be more concerned with the actual non-inflated cost of its decisions. Hence, we gradually warp the logistic function by increasing both h and k after each iteration, as shown in Fig. 5. In the final stages of search, this resembles a step function, encouraging any remaining violations to be eliminated regardless of the wirelength consumed. We found this modification to dramatically improve the success of the routing engine, as compared with the static logistic function.

IV. MAIZEROUTER—ARCHITECTURE FUNDAMENTALS

In the previous section, we focused primarily on the high-level algorithmic design decisions of MAIZEROUTER. However, a separate yet equally important issue that arises in our implementation (and that of any router, for that matter) is that of *model maintenance*. For one to assume that low-level tasks (such as topological reconstruction) can be handled transparently, significant attention must be paid to the internal representations and procedures used to dynamically update solution structure.

In this section, we present the underlying architecture that enables global solution modifications to be performed with minimal complexity and computational expense. As such, these general principles could be applied to any global routing recipe, although we will show that they are particularly well suited to the search space of MAIZEROUTER.

A. Interdependent Net Decomposition

One common attribute of academic global routers is the tendency to decompose Steiner trees into two-pin nets (or wires) and to subsequently route these subnets independently from one another, a process that we will hereafter refer to as *wire-independent* net decomposition. This holds true for the entire flow of *BoxRouter*, which performs no topological reconstruction after its initial Steiner trees have been created and decomposed. *FastRoute* 2.0 is slightly more complex, as it may need to construct entirely new tree topologies from scratch during multisource multitarget maze routing. However, its monotonic algorithm for routing two-pin nets does not consider new decompositions and has the potential to create duplicate (and unnecessary) wires.

Given that our edge manipulation algorithms make heavy use of topological reconstruction (a technique that has been largely avoided by academic global routers due to its complexity and computational expense), we require a means to incrementally modify the position of individual segments without resorting to full-blown tree reconstruction. Such a task is made difficult if trees are maintained in the traditional manner (i.e., using an *explicit* representation of topology), since small local perturbations may introduce large defects in the existing topological construction.

Hence, a key design decision in the development of MAIZEROUTER is an internal representation that performs the following:

1) enables fast incremental updates;
2) allows topology to be modified easily and cheaply;
3) prevents the creation of duplicate or overlapping wires.

We achieve these criteria by way of an *inter*dependent decomposition of routing segments. MAIZEROUTER does not encode net topologies explicitly and instead stores the routing solution for each net simply as an unnested collection of flat wires. For instance, if we designate the upper left coordinate

of Fig. 2(a) as (0, 6), the following set of intervals captures the vertical and horizontal components of its corresponding tree:

VERT. EDGES	HORIZ. EDGES
$[(0,0)-(0,1)]$	$[(0,1)-(4,1)]$
$[(0,4)-(0,5)]$	$[(4,1)-(5,1)]$
$[(\mathbf{4},\mathbf{1})-(\mathbf{4},\mathbf{4})]$	$[(0,4)-(4,4)]$
$[(5,0)-(5,1)]$	

The segment selected to be shifted has been highlighted in bold. The edge shifting procedure chose $x = 8$ as its new location; this choice will result in the removal of

$$[(4,1)-(4,4)]$$

and the addition of

$$\{[(4,1)-(8,1)],[(8,1)-(8,4)],[(4,4)-(8,4)]\}.$$

However, note that the first of these three segments overlaps with an existing subnet (namely, $[(4,1)-(5,1)]$). MAIZEROUTER detects this duplication and responds by transparently decomposing the segment $[(4,1)-(8,1)]$ into two adjacent edges, namely, $[(4,1)-(5,1)]$ and $[(5,1)-(8,1)]$. Only the latter of these two will be inserted into the global set of flat wires

VERT. EDGES	HORIZ. EDGES
$[(0,0)-(0,1)]$	$[(0,1)-(4,1)]$
$[(0,4)-(0,5)]$	$[(4,1)-(5,1)]$
~~$[(4,1)-(4,4)]$~~	$[(0,4)-(4,4)]$
$[(5,0)-(5,1)]$	$+[(\mathbf{5},\mathbf{1})-(\mathbf{8},\mathbf{1})]$
$+[(\mathbf{8},\mathbf{1})-(\mathbf{8},\mathbf{4})]$	$+[(\mathbf{4},\mathbf{4})-(\mathbf{8},\mathbf{4})]$

In general, each segment to be added must be checked against all other segments for duplication and will be broken down into irredundant subcomponents as necessary.

Observe that even though we have made no explicit attempt to modify tree topology, this new set of intervals *implicitly* reflects a different topology than before. Furthermore, since we operate on a segment-by-segment basis, only those edges affected by the local change will be modified to accommodate the candidate path selected by extreme edge shifting. Hence, interdependent net decomposition models the global interaction that occurs between routing segments (in contrast to the model of strict independence assumed in traditional two-pin decomposition), and it also enables cheap incremental updates to this underlying structure.

B. Garbage Collection

Recall from previous sections that our edge manipulation algorithms may, on occasion, produce superfluous wires. For instance, Fig. 2(c) shows a solitary dead-end routing segment on the far right of the tree that could be removed without affecting connectivity. This defect is a consequence of abandoning nonpin Steiner points when shifting a segment across existing wires within the tree.

Fortunately, such edges can be easily removed through a process that we loosely term *garbage collection*, as it eliminates leftover remnants of wasted wire. Fig. 6 shows a rough outline of this procedure. We cycle through all routing segments for a recently modified net and process each endpoint that begins or terminates a segment. Any Steiner node that is seen only once (provided that it is not a pin) is, by definition, a dead end and may be safely removed. The computational complexity is linear in the number of segments.

```
GARBAGE-COLLECTION(Net n_i)
1.  Map(Point → Set⟨Edge⟩) segments
2.  for each edge e in edges(n_i)
3.    segments[e.from].add(e)
4.    segments[e.to].add(e)
5.  for each Steiner node p in nodes(n_i)
6.    if (p ∉ pins(n_i) and segments[p].size() = 1)
7.      removeEdges(segments[p])
```

Fig. 6. Pseudocode for garbage collection.

```
NET-DEFRAGMENT(Net n_i)
1.  Map(Point → Set⟨Edge⟩) segments
2.  for each edge e in edges(n_i)
3.    segments[e.from].add(e)
4.    segments[e.to].add(e)
5.  for each Steiner node p in nodes(n_i)
6.    if (p ∉ pins(n_i) and segments[p] = 2)
7.      if (direction(segments[p].first) = direction(segments[p].last))
8.        removeEdges(segments[p])
9.        addEdge(join(segments[p].first, segments[p].last))
```

Fig. 7. Pseudocode for net defragmentation.

The specific set of intervals corresponding to Fig. 2(c) is as follows:

VERT. EDGES	HORIZ. EDGES
$[(\mathbf{0},\mathbf{0})^*-(0,1)]$	$[(0,1)-(4,1)]$
$[(0,4)-(\mathbf{0},\mathbf{5})^*]$	$[(4,1)-(5,1)]$
$[(\mathbf{5},\mathbf{0})^*-(5,1)]$	$[(0,4)-(4,4)]$
$[(5,1)-(5,2)]$	$[(4,4)-(8,4)]$
$[(\mathbf{8},\mathbf{1})^*-(8,2)]$	$[(5,2)-(8,2)]$
$[(8,2)-(8,4)]$	

Of all the nodes that fall on the endpoints of these segments, only the four marked with $*$ occur exactly once

$$\{(0,0),(0,5),(5,0),(8,1)\}.$$

The first three are pins; the final coordinate lies at the end of our superfluous wire, which should be removed. Fig. 2(d) shows our example after garbage collection has identified and removed the unneeded branch.

We willfully admit that the process of garbage collection is simple and straightforward; nevertheless, it is a necessary and important procedure for removing excess routing segments that arise from our approach to restructuring tree topologies. Refer to Fig. 3(b) for an additional example of unneeded segments eliminated via garbage collection.

C. Net Defragmentation

Occasionally, edge retraction alone may be incapable of collapsing a net into its minimal possible footprint, due to the internal fragmentation of routing segments. This effect is

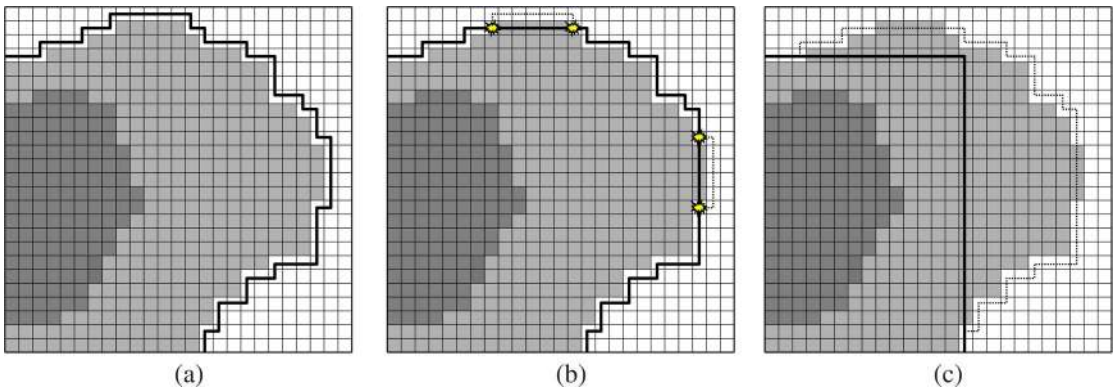

Fig. 8. Illustration of the need for net defragmentation. (a) Maze routing has avoided an area whose routing resources are at near capacity, creating an excessively long detour. (b) Single application of edge retraction collapses two "folds" of the solution but is unable to make further improvements. (c) Desired routing solution, obtained after repeated interleavings of edge retraction and net defragmentation.

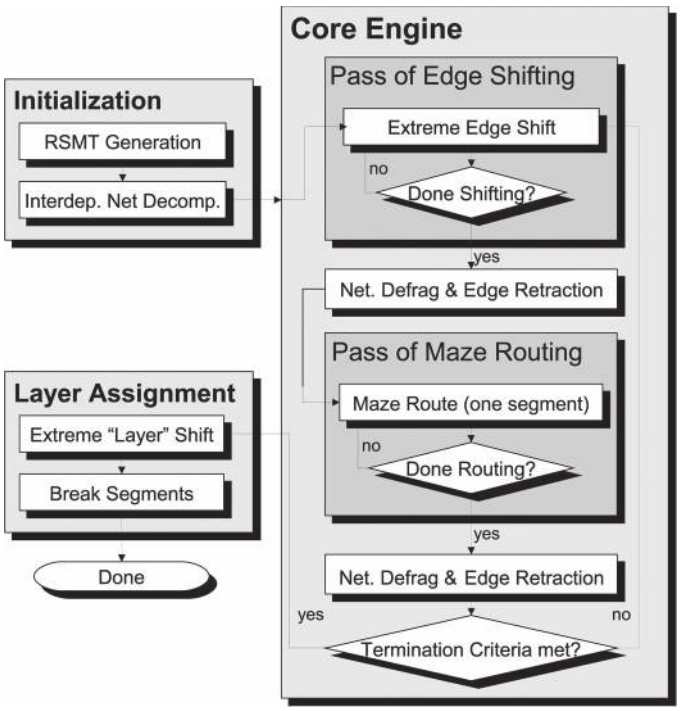

Fig. 9. Complete flow of MAIZEROUTER.

often observed after occasional applications of maze routing, a process that tends to create many bends. Consider Fig. 8(a), in which maze routing has avoided a near-capacity (but not yet at-capacity) area shown in light gray. Once a fully routed solution has been obtained, edge retraction will attempt to condense these segments to reduce wirelength. Fig. 8(b) shows a single application of edge retraction on two "folds" of the solution. Although both of the newly added segments are now adjacent to pairs of neighboring sibling subnets, our internal representation has failed to consolidate these edges into unbroken wires. As a result, no further retractions can be performed in either region, since none of the three individual edges can be slid independently from the other two.

As a result, we introduce a means to rejoin adjacent routing segments through *net defragmentation*. Fig. 7 shows a rough outline of this procedure. As in garbage collection, we cycle

TABLE I
ISPD '98 BENCHMARKS

name	nets	grids	v. cap	h. cap
ibm01	11507	64×64	12	14
ibm02	18429	80×64	22	34
ibm03	21621	80×64	20	30
ibm04	26163	96×64	20	23
ibm05	27777	128×64	42	63
ibm06	33354	128×64	20	33
ibm07	44394	192×64	21	36
ibm08	47944	192×64	21	32
ibm09	50393	256×64	14	28
ibm10	64227	256×64	27	40

through all routing segments for a recently modified net and process each endpoint that begins or terminates a segment. Any Steiner node that lies at the intersection of a single pair of unidirectional routing segments represents an internal rift that may be repaired. The computational complexity is again linear in the number of segments. Fig. 8(c) shows a fully collapsed solution after repeated interleavings of edge retraction and net defragmentation.[5]

Our running example can benefit from net defragmentation as well. In the previous table, the horizontal segments $[(0,1)-(4,1)]$ and $[(4,1)-(5,1)]$ are the only two subnets to converge at the coordinate (4,1) and may thus be combined

VERT. EDGES	HORIZ. EDGES
$[(0,0)-(0,1)]$	$\triangleright\,[(\mathbf{0},\mathbf{1})-(\mathbf{5},\mathbf{1})]\,\triangleleft$
$[(0,4)-(0,5)]$	$[(0,4)-(4,4)]$
$[(5,0)-(5,1)]$	$[(4,4)-(8,4)]$
$[(5,1)-(5,2)]$	$[(5,2)-(8,2)]$
$[(8,2)-(8,4)]$	

In this case, consolidation does not serve to enable edge retraction, but it may aid future applications of edge shifting. For instance, this larger segment may now be shifted downward as a whole one unit to completely avoid the congested region, whereas an attempt to shift either of the original edges

[5]The use of defragmentation within the context of interdependent net decomposition parallels recent techniques for the consolidation of edges within explicitly defined Steiner tree topologies [18].

TABLE II
COMPARISON OF *FastRoute* 2.0, *BoxRouter*, AND MAIZEROUTER ON THE ISPD '98 BENCHMARKS

	FastRoute 2.0			BoxRouter			MAIZEROUTER (our work)			vs. *FR 2.0*		vs. BoxRouter	
name	ovfl	wlen	cpu(s)	ovfl	wlen	cpu(s)	ovfl	wlen	cpu(s)	wlen	cpu(s)	wlen	cpu(s)
ibm01	31	68489	0.72	102	65588	6.46	**0**	63720	3.88	−7.55%	5.01×	−2.93%	0.69×
ibm02	0	178868	0.93	33	178759	24.82	**0**	170342	3.40	−4.63%	3.90×	−4.94%	0.17×
ibm03	0	150393	0.60	0	151299	13.58	**0**	147078	2.83	−1.64%	4.35×	−2.87%	0.22×
ibm04	64	175037	1.88	309	173289	18.99	**0**	170095	19.78	−2.69%	9.01×	−1.88%	1.17×
ibm05	—	—	—	0	409747	40.01	**0**	410031	1.40			+0.07%	0.04×
ibm06	0	284935	1.36	0	282325	27.25	**0**	279566	5.78	−1.66%	3.45×	−0.99%	0.20×
ibm07	0	375185	1.60	53	378876	42.55	**0**	369340	6.11	−1.42%	3.04×	−2.58%	0.14×
ibm08	0	411703	2.36	0	415025	71.28	**0**	406349	8.98	−1.23%	2.86×	−2.14%	0.11×
ibm09	3	424949	1.92	0	418615	55.05	**0**	415852	8.14	−2.21%	3.31×	−0.66%	0.13×
ibm10	0	595622	2.79	0	593186	80.63	**0**	585921	9.70	−1.59%	3.45×	−1.24%	0.14×
							average			−2.92%	4.27×	−2.02%	0.30×

individually would only partially reduce congestion [in addition to creating a new segment emanating from (4,1)].

D. Layer Assignment in Multilayer Routing

The basic components of our routing algorithm extend easily to the case of 3-D routing. First, MAIZEROUTER projects all pins, capacities, blockages, etc., into single-layer routing nets in two dimensions. It then iteratively unfolds this solution level by level, using the same strategies to achieve layer assignment as it did to avoid congestion in the original projected solution. For instance, the parallel segments of extreme edge shifting correspond to *vias* in the 3-D solution, and candidate positions for the central segment correspond to candidate *layers*.

Provided that there is no limit on the number of vias between the cells of any pair of layers (as was the case in the Global Routing Contest), segments may be broken as needed to obtain a solution whose overflow is identical to that in the projected solution.

E. Complete Flow of MAIZEROUTER

In Fig. 9, we provide the complete high-level flow of MAIZEROUTER. After initialization (including RSMT generation and interdependent net decomposition), extreme edge shifting is repeatedly applied over all segments identified for relocation. Afterward, net defragmentation and edge retraction are performed to reclaim routing resources, followed by a pass of maze routing and additional recovery. This general process continues until a user-specified set of termination criteria is met; for instance, until zero overflow (or more generally, a particular overflow threshold) has been reached, or a time-out limit has passed. Finally, the projected routes are extracted into a multilayer solution (using conservative edge shifting if possible but breaking segments as needed).

Although we found this particular recipe to be successful, several other combinations and orderings of techniques exist. The development of metaalgorithms to control this flow is worthy of continued research but was not a primary focus of our study.

We stress that this flow diagram does not (and cannot) capture all the salient details of MAIZEROUTER. Critically important lower level operations that form our incremental routing "maintenance system" (such as interdependent net decomposition, garbage selection, etc.) are not shown but are selectively invoked when routing structures are modified.

TABLE III
ISPD '07 BENCHMARKS

name	nets	grids	v. cap	h. cap
adaptec1	219794	324 × 324	70	70
adaptec2	260159	424 × 424	80	80
adaptec3	466295	774 × 779	62	62
adaptec4	515304	774 × 779	62	62
adaptec5	867411	465 × 468	110	110
newblue1	331663	399 × 399	62	62
newblue2	463213	557 × 463	110	110
newblue3	551667	973 × 1256	80	80

V. EXPERIMENTAL RESULTS

A. Results on the ISPD '98 Benchmarks

In Table I, we provide a summary of the original ten ISPD '98 benchmarks [31] that have become standard in the routing literature. To empirically evaluate the performance of MAIZEROUTER on these instances, we compare it with the *FastRoute* 2.0 [23] and the original *BoxRouter* [5], the two leading global routers prior to the release of the larger instances.

In Table II, we present the results of MAIZEROUTER against *BoxRouter* and *FastRoute* 2.0. For each test case, we report the total number of overflows, total wirelength, and CPU runtime for each solver. All experiments were performed on a 64-b machine with 16 GB of memory and a dual-core 2.8-GHz AMD Opteron(tm) Processor.[6] We confirm that *BoxRouter* completes six of the ten instances with wirelengths comparable with those of the *Chi* dispersion router [10] and that *FastRoute* 2.0 fails to route three of the benchmarks. However, MAIZEROUTER outperforms *BoxRouter* on all counts, successfully routing all benchmarks, reducing wirelength by an average of 1.84%, and running significantly faster over the entire set of problems. As compared with *FastRoute* 2.0, MAIZEROUTER tends to be slower but improves upon wirelength by 2.73%.

B. Results on the ISPD '07 Benchmarks

Table III summarizes key statistics of the test cases released during the Global Routing Contest. The relative sizes of these

[6]At the time of this writing, a binary of *FastRoute* has not yet been made publicly available, and thus, we report the runtime statistics from [23].

TABLE IV
COMPARISON OF MAIZEROUTER AGAINST MATURED MULTILAYER GLOBAL ROUTERS ON THE ISPD '07 BENCHMARKS

		FGR [25]			BoxRouter 2.0 [4]			Archer [21]			NTHU-Route [8]			MAIZEROUTER		
	name	ovfl	wlen	cpu(m)	ovfl	wlen	cpu(m)	ovfl	wlen	cpu(m)	ovfl	wlen	cpu(m)	ovfl	wlen	cpu(m)
3-D	adaptec1	0	88.45	430	0	92.04	N/A	0	113.80	87	0	90.56	94	0	93.79	223
	adaptec2	0	89.89	64	0	94.28		0	112.56	23	0	92.17	17	0	91.65	42
	adaptec3	0	199.66	243	0	207.41		0	244.08	51	0	205.04	65	0	202.13	83
	adaptec4	0	182.96	83	0	186.42		0	221.57	12	0	188.43	10	0	185.68	14
	adaptec5	0	259.98	740	0	270.41		0	334.09	248	0	265.03	268	0	265.90	339
	newblue1	452	94.26	1442	394	92.94		682	116.98	50	352	90.91	38	1194	95.27	87
	newblue2	0	132.16	18	0	134.64		0	166.50	7	0	136.01	4	0	133.48	9
	newblue3	38580	173.71	1501	38958	172.44		33394	198.77	163	31800	168.40	358	32825	177.44	182
2-D	adaptec1	0	54.44	451	0	58.37		0	58.27	86	0	57.11	93	0	55.91	221
	adaptec2	0	52.30	56	0	55.69		0	54.60	22	0	54.46	16	0	53.22	40
	adaptec3	0	130.89	179	0	137.96		0	135.42	49	0	137.16	63	0	133.49	80
	adaptec4	0	125.00	19	0	127.79		0	126.26	11	0	128.66	9	0	124.95	13
	adaptec5	0	152.13	713	0	162.11		0	162.49	246	0	160.30	266	0	157.50	337
	newblue1	452	47.42	1441	400	51.13		682	49.30	49	352	47.78	38	1194	46.97	84
	newblue2	0	76.51	4	0	78.68		0	77.91	6	0	79.22	3	0	76.48	7
	newblue3	38580	109.23	1555	38958	111.61		33394	109.25	162	31800	111.00	356	32825	110.56	180
totals		1.147	0.982	4.605	1.157	1.015		1.001	1.138	0.655	0.945	1.004	0.875	1.000	1.000	1.000

benchmarks significantly dwarf those of the older set, both in terms of the number of nets (which has increased by a full order of magnitude), and the scale of the routing grids.

As mentioned earlier, MAIZEROUTER took first place in the 3-D track of the contest, followed closely by *BoxRouter* and FGR.[7] However, in the months following the contest, all three leading routers from the contest matured significantly. In Table IV, we report the latest results of FGR, *BoxRouter* 2.0, and MAIZEROUTER on the ISPD '07 benchmarks. FGR now successfully routes six of the eight 3-D benchmarks, whereas it previously routed only three. The wirelengths of *BoxRouter* 2.0 have been improved as compared with its contest results, although no runtime statistics have yet been published. Two new global routers, named *Archer*and *NTHU-Route*, are also shown in the table and demonstrate particularly impressive runtime performance.

The latest results of MAIZEROUTER are shown in the final columns of Table IV. Overall, its wirelength over all benchmarks has improved considerably (by an average of over 6% as compared with the contest version), and it is now capable of successfully routing `adaptec5`. Despite its relatively simple design, MAIZEROUTER remains within a few percent of the best known results and is significantly faster than the only router with less wire length (FGR), particularly in the unroutable cases that are of great interest to placers. Compared with the high-speed routers *Archer* and *FastRoute*, MAIZEROUTER exhibits better wirelength, and it is also generally more effective than *BoxRouter* 2.0.

VI. FUTURE WORK

Development of MAIZEROUTER is an ongoing process, and we suspect that there remains further room for improvement. In an effort to encourage others to improve upon our results, we have released the source code of MAIZEROUTER to the academic community under a general public license [37].

[7]We omit the details of the contest results here, since they have become largely out of date—see [20] for a complete summary.

VII. CONCLUSION

In this paper, we have presented the complete design and architectural details of MAIZEROUTER, a novel and state-of-the-art global routing engine. MAIZEROUTER reflects a significant leap in progress over previously available academic routing tools, due in part to its simple yet powerful edge-based operations (*extreme edge shifting* and *edge retraction*) and also due to its use of an *interdependent* form of net decomposition, a representation that improves upon traditional two-pin net decomposition by preventing duplication of routing resources and enabling cheap topological reconstruction. The mechanisms of *garbage collection* and *net defragmentation* complement our edge-based operations by eliminating the rifts and leftover routing segments that they produce. We believe that many of our techniques can be incorporated into existing routers to substantially improve their performance and quality.

ACKNOWLEDGMENT

The author would like to thank Prof. I. L. Markov of the University of Michigan for valuable insight and several useful discussions. The author is currently supported by the 2007 IBM Josef Raviv Memorial Postdoctoral Fellowship.

REFERENCES

[1] C. Albrecht, "Global routing by new approximation algorithms for multicommodity flow," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 5, pp. 622–632, May 2001.
[2] Z. Cao, T. Jing, J. Xiong, Y. Hu, L. He, and X. Hong, "DpRouter: A fast and accurate dynamic-pattern-based global routing algorithm," in *Proc. ASP-DAC*, pp. 256–261.
[3] R. C. Carden, IV, J. Li, and C.-K. Cheng, "A global router with a theoretical bound on the optimal solution," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 2, pp. 208–216, Feb. 1996.
[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. ICCAD*, 2007, pp. 503–508.
[5] M. Cho and D. Z. Pan, "BoxRouter: A new global router based on box expansion and progressive ILP," in *Proc. DAC*, 2006, pp. 373–378.
[6] C. Chu, "FLUTE: Fast lookup table based wirelength estimation technique," in *Proc. ICCAD*, 2004, pp. 696–701.

[7] C. C. N. Chu and Y.-C. Wong, "Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design," in *Proc. ISPD*, 2005, pp. 28–35.
[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] R. Goering, *IC Routing Contest Boosts CAD Research*, 2007. [Online]. Available: www.eetimes.com/showArticle.jhtml?articleID=198500084
[10] R. T. Hadsell and P. H. Madden, "Improved global routing through congestion estimation," in *Proc. DAC*, 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, Mar. 1966.
[12] J. Hu and S. S. Sapatnekar, "A survey on multi-net global routing for integrated circuits," *Integr. VLSI J.*, vol. 31, no. 1, pp. 1–49, Nov. 2001.
[13] A. Kahng, I. Mandoiu, and A. Zelikovsky, "Highly scalable algorithms for rectilinear and octilinear Steiner trees," in *Proc. ASP-DAC*, 2003, pp. 827–833.
[14] A. B. Kahng and X. Xu, "Accurate pseudo-constructive wirelength and congestion estimation," in *Proc. SLIP*, 2003, pp. 61–68.
[15] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Predictable routing," in *Proc. ICCAD*, 2000, pp. 110–113.
[16] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Pattern routing: Use and theory for increasing predictability and avoiding coupling," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 7, pp. 777–790, Jul. 2002.
[17] Z. Li, C. J. Alpert, S. T. Quay, S. S. Sapatnekar, and W. Shi, "Probabilistic congestion prediction with partial blockages," in *Proc. ISQED*, 2007, pp. 841–846.
[18] C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, "Efficient obstacle-avoiding rectilinear Steiner tree construction," in *Proc. ISPD*, 2007, pp. 127–134.
[19] J. Lou, S. Krishnamoorthy, and H. S. Sheng, "Estimating routing congestion using probabilistic analysis," in *Proc. ISPD*, 2001, pp. 112–117.
[20] M. D. Moffitt, "MaizeRouter: Engineering an effective global router," in *Proc. ASP-DAC*, 2008, pp. 226–231.
[21] M. M. Ozdal and M. D. F. Wong, "Archer: A history-driven global routing algorithm," in *Proc. ICCAD*, 2007, pp. 488–495.
[22] M. Pan and C. Chu, "FastRoute: A step to integrate global routing into placement," in *Proc. ICCAD*, 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] M. Pan and C. Chu, "IPR: An integrated placement and routing algorithm," in *Proc. DAC*, 2007, pp. 59–62.
[25] J. Roy and I. L. Markov, "High-performance routing at the nanometer scale," in *Proc. ICCAD*, 2007, pp. 496–502.
[26] *Electronic Design Automation for Integrated Circuits Handbook*, L. K. Scheffer, L. Lavagno, and G. Martin, Eds. Boca Raton, FL: CRC Press, 2006, ch. 8: Routing.
[27] C.-W. Sham and E. F. Y. Young, "Congestion prediction in early stages," in *Proc. SLIP*, 2005, pp. 91–98.
[28] J. Westra, C. Bartels, and P. Groeneveld, "Probabilistic congestion prediction," in *Proc. ISPD*, 2004, pp. 204–209.
[29] J. Westra and P. Groeneveld, "Is probabilistic congestion estimation worthwhile?" in *Proc. SLIP*, 2005, pp. 99–106.
[30] [Online]. Available: www.ispd.cc/ispd07_contest.html
[31] [Online]. Available: www.ece.ucsb.edu/~kastner/labyrinth/
[32] [Online]. Available: vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/FastSteiner/
[33] [Online]. Available: home.eng.iastate.edu/~cnchu/flute.html
[34] [Online]. Available: www.cerc.utexas.edu/~thyeros/boxrouter/boxrouter.htm
[35] [Online]. Available: www.cerc.utexas.edu/utda/download/BoxRouter.htm
[36] [Online]. Available: vlsicad.eecs.umich.edu/BK/FGR/
[37] [Online]. Available: www.ispd.cc/ispd07_contest.html

Michael D. Moffitt received the B.S. degree in computer science and engineering, dual M.S. degrees in computer science and operations research, and the Ph.D. degree from the University of Michigan, Ann Arbor, in 2003, 2005, 2006, and 2007, respectively.

He is currently with the Design Productivity Group, IBM Austin Research Laboratory, Austin, TX. His primary research interests include constraint satisfaction, combinatorial optimization, and their application to large-scale problems commonly found in artificial intelligence and computer-aided design.

Dr. Moffit was the recipient of the 2007 Josef Raviv Memorial Postdoctoral Fellowship.