MAIZEROUTER: Engineering an Effective Global Router

Michael D. Moffitt*
IBM Austin Research Lab
11501 Burnet Road, Austin, TX 78758
mdmoffitt@us.ibm.com

Abstract — In this paper, we present MAIZEROUTER, winner of the inaugural Global Routing Contest hosted at ISPD 2007. MAIZEROUTER reflects a significant leap in progress over existing publicly-available routing tools, and abandons popular algorithms such as multicommodity flow-based techniques, ILP formulations, and congestion-driven Steiner tree generation. Instead, the foundation of our algorithm draws upon simple yet powerful edge-based 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. 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. A maintenance mechanism, named *garbage collection*, is introduced to eliminate leftover routing segments. 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.

I. INTRODUCTION

GLOBAL ROUTING is a critical step in modern VLSI physical design. Its importance has recently been cast into the limelight with the CEDA-sponsored Global Routing Contest hosted at ISPD 2007 [25], a competition that attracted nearly a dozen academic and industrial participants from around the globe. As reported in EE Times [7], contests such as these serve as an important bellwether of urgent problems in EDA, 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 [10] to flow-based techniques [1] to Integer-Linear Programming formulations [4] to congestion-driven Steiner tree generation [18]. 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 represent 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 fundamental design and architectural details of MAIZEROUTER, a novel state-of-the-art global routing algorithm. A preliminary version of MAIZEROUTER took 1^{st} Place in the three-dimensional track of the ISPD 2007 Global Routing Contest, outperforming established engines such as *BoxRouter* [4] and *FastRoute* [19], in addition to several newcomers. 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 [18] 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.

These algorithmic contributions are situated atop 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. We also introduce a maintenance mechanism named **garbage collection**, a process whereby leftover routing segments produced by our edge-based operations are removed from the solution. Collectively, these operations permit a broad search space that previous algorithms have been unable to achieve. Combined with a moderate amount of 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 over $7.5\times$ faster than *BoxRouter* and producing 2.02% shorter wirelength. In addition, a reduction in wirelength of 2.92% is observed over *FastRoute* 2.0.

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, MAIZEROUTER, and its principle algorithmic components. In Section IV, we discuss the underlying representation that MAIZEROUTER uses to encode and manipulate its routing solutions, along with some basic routines to maintain 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

*This material is based, in part, on preliminary work performed by the author at the University of Michigan's Artificial Intelligence Laboratory.

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

edges E. As shown in Figure 1, each vertex $v_i \in V$ corresponds to a particular rectangular region (or cell) of the chip, 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 [14]. 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 three-dimensional routing, this calculation can also include *vias*, special wires used to connect routing segments between consecutive layers of metal. Finally, one may be concerned with the *runtime* needed to construct the solution. This is especially true in cases where global routing is repeatedly used to guide a placement algorithm [20], as is typical in "V-shaped" coarsening and refinement strategies. Global routing is a textbook example of a multi-objective optimization problem, in which the relative importance of the individual criterion depend heavily on the context in question.

B. Global Routing: Basic Algorithms

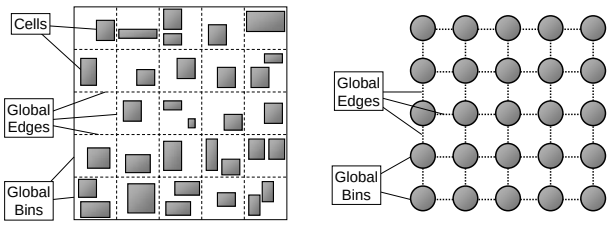
As is the case with many large-scale optimization problems, a wide variety of algorithms have been proposed for global routing, many of which are discussed in a comprehensive survey on the topic [10]. *Maze routing* is a grid-based search algorithm that has long held a reputation as a brute-force approach 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 BFS or Dijkstra's algorithm, whereas A*-style approaches (using, for instance, the Manhattan distance between two points as an admissible heuristic) often perform substantially better.

Pattern routing [14] 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 (local) 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 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 [3, 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* [12, 17, 22, 23] 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 [24], it is still commonly used in industrial placement tools [15].

C. Global Routing: Leading Solvers

Prior to 2006, the leading global routing tools were Labyrinth [13] (which uses primarily pattern routing) and the *Chi* Dispersion Router [8] (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.



(a) Global bin decomposition (b) Corresponding grid graph

Fig. 1. The bin decomposition and grid graph of the global routing problem formulation.

BoxRouter [4] progressively expands a box initiated from the most congested region of the chip, applying an integer linear programming (ILP) formulation to re-route 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 [18] uses a congestion map to warp the structure of a Hanan grid [9] during Steiner tree generation, followed by edge shifting and a form of pattern routing. It has also recently been enhanced with monotonic routing and multi-source multi-destination maze routing [19], although the most recent version of *FastRoute* did not prove to be competitive in the competition.

Both *BoxRouter* and *FastRoute* make use of the publicly-available *FLUTE* package [5, 6, 28] 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 to handle nets of larger size.

III. MAIZEROUTER – ALGORITHM FUNDAMENTALS

In this section, we describe the algorithmic details and overall flow of our routing engine, named MAIZEROUTER.

A. Solution Initialization

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 RSMT or RMST package (such as FastSteiner [11, 27]) 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.

B. Extreme Edge Shifting

Of the many techniques that the engine *FastRoute* [18] 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 Figure 3(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 re-route the path between a pair of points by exploiting the presence of neighboring wires.

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

```
EXTREME-EDGE-SHIFT(Edge e, int window)
  // vertical edge case (e.from.x = e.to.x)
1.  Point top = e.from, bottom = e.to
2.  int minX = max(0, top.x – window)
3.  int maxX = min(grids.W, top.x + window)
4.  int bestCost = 0, bestVal = top.x

5.  for y = [top.y .. bottom.y]
6.    bestCost += cost(demandV[top.x][y] – 1, capV)

7.  shiftCost = 0 // cost accumulates as edge shifts left
8.  for x = [x–1 .. minX]
9.    unless (netsH[x][top.y].contains(e.net))
10.     shiftCost += cost(demandH[x][top.y], capH)
11.   unless (netsH[x][bottom.y].contains(e.net))
12.     shiftCost += cost(demandH[x][bottom.y], capH)
13.   cost = shiftCost
14.   for y = [top.y .. bottom.y]
15.     unless (netsV[x][y].contains(e.net))
16.       cost += cost(demandV[x][y], capV)
17.   if (cost < bestCost) bestCost = cost, bestVal = x
18. shiftCost = 0 // cost accumulates as edge shifts right
19. for x = [x+1 .. maxX]
20.   ... // repeat lines 9 through 17

21. Point temp1(bestVal, top.y), temp2(bestVal, bottom.y)
22. removeEdge(e)
23. addEdge(Edge(top, temp1)) // top of 'C'
24. addEdge(Edge(temp1, temp2)) // side of 'C'
25. addEdge(Edge(temp2, bottom)) // bottom of 'C'
```

Fig. 2. Pseudocode for Extreme Edge Shifting

Although powerful, edge shifting is sharply limited in scope, in that it provides a relatively narrow band 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 tree topology be preserved when moving an existing segment out of a congested region. 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, Figure 3(b) demonstrates a case where the bold edge from Figure 3(a) has been moved to the far right, a region where routing resource is relatively underutilized. Two *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 depicted in Figure 3(c), this time in an upward direction.

Pseudocode for our extreme edge shifting procedure is provided in Figure 2. The algorithm is given a particular edge e, and the range of the sweep is parameterized by the variable *window*. As different locations for the edge are attempted, the algorithm computes the cost of the move incrementally, allowing $O(l \times window)$ runtime (where l is the length of the edge). One may choose from any number of strategies for *cost*() (i.e., step, linear, etc.) although our implementation makes use of a logistic function whose shape is dynamically adjusted during search. Just as in traditional edge shifting, cost need not be accumulated for any cell that contains a wire for the current net, hence we check for such a condition in lines 9, 11, and 15. We also note briefly that our implementation of *addEdge*() transparently decomposes the given segment into several irredundant wires if there does happen to be overlap (that is, with other segments of the same 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

```
SHIFT-EDGES(float thresh, float window)
  // vertical edge case (e.from.x = e.to.x)
1.  for x = [0 .. grids.W]
2.    for y = [0 .. grids.H]
3.      if (demandV[x][y] > thresh × capV)
4.        for each Edge e in edgesV[x][y]
5.          EXTREME-EDGE-SHIFT(e, window)
```

Fig. 4. Pseudocode for a Pass of Extreme Edge Shifting

threshold, it will attempt to detour as many segments away from that cell as possible. Figure 4 demonstrates this high-level procedure. 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 re-route the entire tree of any net (as would typically be done in Rip-up-and-re-route). 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 Figure 3(c). We will address this concern in Section IV-B when discussing 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* [4], since both reduce wirelength as a postprocessing step (although our approach avoids the use of maze routing).

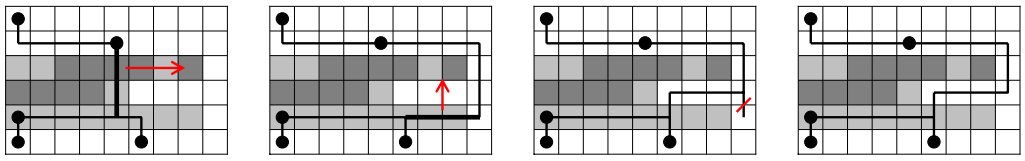
In Figure 5, we demonstrate edge retraction on our running example. The segment on the far right can be moved two units to the left without incurring additional overflow (Figure 5(a)). After this translation is performed, dead-end segments are detected and eliminated (Figure 5(b)), thereby fulfilling the promise of reduced wirelength. The final routing for this net is shown in Figure 5(c).

IV. MAIZEROUTER – ARCHITECTURE FUNDAMENTALS

In the previous section, we focused on the high-level algorithmic design decisions of MAIZEROUTER, assuming that low-level maintenance tasks (such as topological reconstruction) could be handled transparently. In this section, we briefly discuss the underlying architecture that allows these operations to be performed with minimal complexity and computational expense.

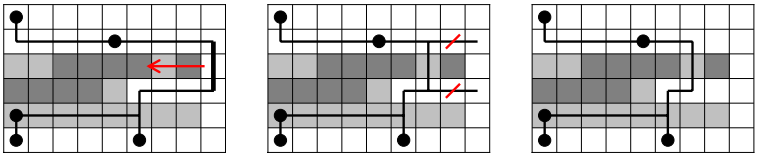
[2] A form of 'U'-shaped move is proposed in [16], but it too is constrained by the predetermined topology of the net.

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



(a) (b) (c) (d)

Fig. 3. Two applications of *extreme edge shifting* followed by *garbage collection*. Dark grey indicates an area of overflow; light grey indicates an at- or near-capacity area; white indicates an area of relatively low demand.



(a) (b) (c)

Fig. 5. An example of *edge retraction* followed by *garbage collection*. Dark grey indicates an area of overflow; light grey indicates an at- or near-capacity area; white indicates an area of relatively low demand.

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 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 multi-source multi-target 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.

Conventional wisdom stipulates that during the entire course of a routing procedure, the topology of a net may alternatively be understood and represented recursively as a tree. However, 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.

In understanding the search space of MAIZEROUTER, it is more useful to imagine the routing of a net simply as a global, unnested collection of intervals (or flat wires) in two-dimensional (or, for multi-layer routing, three-dimensional) space. In other words, we do not explicitly encode net topology, as we instead operate on segments obtained from an *inter*dependent decomposition of routing segments, in which the global set of non-overlapping intervals is shared among all subnets. This facilitates our search paradigm, where flat wires are individually re-routed to reduce (or eliminate) overflow while preserving connectivity of their corresponding nets. For instance, if we designate the upper-left coordinate of Figure 3(a) as $(0, 6)$, the following set of intervals captures the vertical and horizontal components of the tree:

VERT. EDGES	HORIZ. EDGES
$[(0,0)-(0,1)]$	$[(0,1)-(4,1)]$
$[(0,4)-(0,5)]$	$[(4,1)-(5,1)]$
$[(4,1)-(4,4)]$	$[(0,4)-(4,4)]$
$[(5,0)-(5,1)]$	

Observe that even though we have made no explicit attempt to encode tree topology, this set of intervals *implicitly* reflects the overall structure of the net and all necessary branching points. Furthermore, since we operate on a segment-by-segment basis, only those edges affected by local changes must be modified to accommodate the candidate paths selected by our edge-based operations.[4] 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 also enables cheap incremental updates.

B. Garbage Collection

Recall from previous sections that our edge manipulation algorithms may, on occasion, produce superfluous wires. For instance, in Figure 3(c) we observed a solitary dead-end routing segment that could be removed without affecting connectivity. This defect is a consequence of abandoning non-pin nodes that are rendered unnecessary when shifting a segment across existing wires within the tree.

Fortunately, such edges can be easily removed through a process we loosely term *garbage collection*, as it eliminates leftover remnants of wasted wire. Figure 6 provides 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 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.

The process of garbage collection is simple and straightforward; nevertheless, it a necessary and important procedure for removing excess routing segments that arise from our approach to restructuring tree topologies. Figure 3(d) shows our example after garbage collection has identified and removed the unneeded branch.

[4] MAIZEROUTER uses several techniques to accomodate the dynamic addition, removal, and merging of segments; due to space limitations, we omit such details here.

```
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 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

name	nets	grids
ibm01	11507	64 × 64
ibm02	18429	80 × 64
ibm03	21621	80 × 64
ibm04	26163	96 × 64
ibm05	27777	128 × 64
ibm06	33354	128 × 64
ibm07	44394	192 × 64
ibm08	47944	192 × 64
ibm09	50393	256 × 64
ibm10	64227	256 × 64

TABLE I
THE ISPD '98 BENCHMARKS

name	nets	grids
adaptec1	219794	324 × 324
adaptec2	260159	424 × 424
adaptec3	466295	774 × 779
adaptec4	515304	774 × 779
adaptec5	867411	465 × 468
newblue1	331663	399 × 399
newblue2	463213	557 × 463
newblue3	551667	973 × 1256

TABLE II
THE ISPD '07 BENCHMARKS

	FastRoute 2.0		MAIZEROUTER		Imprv.
name	ovfl	wlen	ovfl	wlen	(%)
ibm01	31	68489	**0**	63720	−7.48%
ibm02	0	178868	**0**	170342	−5.01%
ibm03	0	150393	**0**	147078	−2.25%
ibm04	64	175037	**0**	170095	−2.91%
ibm06	0	284935	**0**	279566	−1.92%
ibm07	0	375185	**0**	369340	−1.58%
ibm08	0	411703	**0**	406349	−1.32%
ibm09	3	424949	**0**	415852	−2.19%
ibm10	0	595622	**0**	585921	−1.66%
			average		−2.92%

TABLE IV
COMPARISON OF *FastRoute* 2.0 AND MAIZEROUTER

C. Layer Assignment in Multi-layer Routing

The basic components of our routing representation and algorithms extend easily to the case of three-dimensional routing. First, MAIZEROUTER projects all pins, capacities, blockages, etc. into a 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 3D 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.

V. EXPERIMENTAL RESULTS

A. Results on ISPD '98 Benchmarks

In Table I we provide a summary of the ten ISPD '98 benchmarks [26] that have become standard in the routing literature. To empirically evaluate the performance of MAIZEROUTER on these instances, we compare it to *FastRoute* 2.0, *BoxRouter* [29], and the *Chi* dispersion router deployed in Fengshui 5.1.

In Table III we present the results of MAIZEROUTER against *BoxRouter* and *Chi*. For each test case, we report the total number of overflows, total wirelength, and CPU runtime for each solver (with all tests being performed on the same machine). We confirm that *BoxRouter* is indeed substantially superior to *Chi* both in overflow and runtime, one of the reasons for its reputation as a highly robust router. It completes six of the ten instances, and also consistently produces comparable wirelengths. However, one can observe that MAIZEROUTER outperforms *BoxRouter* on all counts, successfully routing the four remaining benchmarks, reducing wirelength by an average of 2.02%, and running over 7.5× faster on the entire set of problems.

A comparison of solution quality between MAIZEROUTER and the most recent results of *FastRoute* 2.0 [19] is shown in Table IV.[5] *FastRoute* 2.0 fails to route three of the benchmarks, and its wirelength is 2.92% worse on average than that of MAIZEROUTER.

B. Results of the ISPD 2007 Global Routing Contest

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

In Table V, we provide raw statistics for the solutions of all entries to the routing competition; as can be seen, these submissions span a remarkably wide range in quality. Solutions for each instance were ranked from best to worst by total overflow (using maximum overflow and wirelength as tie-breakers), and the average ranking for each router was taken as a final score.

In the 3D category, MAIZEROUTER (1^{st} Place) and *BoxRouter* (2^{nd} Place) were the only two contestants to produce routable solutions for at least half of the benchmarks. That being said, the typical number of violations produced by FGR (3^{rd} Place) for routable instances was often very close to zero. However, despite their infrequency, these small overflows prevented FGR from playing a much more competitive role in the 3D category.[6] Four of the six remaining teams were incapable of routing more than a single instance, with one router in particular (Bockenem) completing none. Notably, *BoxRouter* generated one more fully-routed solution (adaptec5) than could MAIZEROUTER. In the 2D category, FGR (1^{st} Place) displayed an extremely impressive showing, fully routing many solutions with significantly lower wirelength than other entries (including our own).

VI. FUTURE WORK

Development of MAIZEROUTER is an ongoing process, and despite its victory at the ISPD 2007 Global Routing Competition, we suspect that there remains further room for improvement. For instance, its loss to FGR in the two-dimensional track suggests that there are likely additional optimizations that could significantly enhance the solution quality of MAIZEROUTER. Furthermore, the ability of *BoxRouter* to route one more contest benchmark than MAIZEROUTER points to yet another area of where progress can (and should) be made. 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 [30].

VII. CONCLUSION

In this paper, we have presented the design and architectural details of MAIZEROUTER, a novel and state-of-the-art global routing engine. MAIZEROUTER reflects a significant leap in progress over existing publicly-available 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 mechanism of *garbage collection* complements our edge-based operations by eliminating the leftover routing segments they produce. We believe that many of our techniques can be incorporated into existing routers to substantially improve their performance and quality, as evidenced by our success at ISPD 2007.

[5] At the time of this writing, a binary of *FastRoute* has not yet been made publicly available, and so we are unable to provide a runtime comparison.

[6] An upcoming version of the FGR solver is expected to correct this issue [21].

	Chi Dispersion Router			BoxRouter			MAIZEROUTER (our work)			Imprv. on *Chi*		Imprv. on BoxRouter	
name	ovfl	wlen	cpu(s)	ovfl	wlen	cpu(s)	ovfl	wlen	cpu(s)	wlen	cpu(s)	wlen	cpu(s)
ibm01	66006	189	10.28	102	65588	5.23	**0**	63720	3.61	−3.59%	2.85×	−2.93%	1.45×
ibm02	178892	64	32.62	33	178759	20.83	**0**	170342	3.63	−5.02%	8.99×	−4.94%	5.74×
ibm03	152392	10	23.97	0	151299	11.77	**0**	147078	2.61	−3.61%	9.19×	−2.87%	4.51×
ibm04	173241	465	36.84	309	173289	14.46	**0**	170095	16.94	−1.85%	2.18×	−1.88%	0.85×
ibm05	412197	0	71.38	0	409747	36.73	**0**	410031	1.37	−0.53%	52.10×	+0.07%	26.81×
ibm06	289276	35	54.56	0	282325	23.22	**0**	279566	4.70	−3.47%	11.61×	−0.99%	4.94×
ibm07	378994	309	83.23	53	378876	35.87	**0**	369340	4.86	−2.61%	17.13×	−2.58%	7.38×
ibm08	415285	74	77.51	0	415025	59.67	**0**	406349	6.74	−2.20%	11.50×	−2.14%	8.85×
ibm09	427556	52	85.21	0	418615	47.82	**0**	415852	6.35	−2.81%	13.42×	−0.66%	7.53×
ibm10	599937	51	145.01	0	593186	69.39	**0**	585921	9.64	−2.39%	15.04×	−1.24%	7.20×
								average		−2.81%	14.40×	−2.02%	7.53×

TABLE III
COMPARISON OF *Chi* DISPERSION ROUTER, *BoxRouter*, AND MAIZEROUTER

		adaptec1		adaptec2		adaptec3		adaptec4		adaptec5		newblue1		newblue2		newblue3	
	Router	ovfl	wlen	ovfl	wlen	ovfl	wlen	ovfl	wlen	ovfl	wlen	ovfl	wlen	ovfl	wlen	ovfl	wlen
3-D	1. MaizeRouter	0	100	0	98	0	214	0	194	2	305	1348	102	0	140	32840	184
	2. BoxRouter	0	104	0	103	0	236	0	212	0	298	400	102	0	155	38976	196
	3. FGR	60	91	50	92	0	203	0	186	2480	265	2668	93	0	136	53648	168
	4. FastRoute	122	249	500	244	0	523	0	469	9894	708	2602	248	0	380	34236	443
	5. NTHU-R(3)	3476	194	3588	177	64	406	0	303	20632	505	5526	180	0	232	38146	317
	6. FlexRouter	8698	120	7370	114	950	269	18	227	21802	336	7636	111	0	171	39488	216
	7. Bockenem	1240	254	10428	211	166498	407	7370	392	98950	576	3936	220	674	272	301052	309
	8. NTU1-R(9)	62638	115	24738	112	31178	413	1342	252	208804	556	17872	115	0	168	148646	203
	9. NTU2-R(13)*	32488	253	13662	243	43332	668	4064	600	120602	719	6570	200	0	362	64102	605
2-D	1. FGR	0	56	0	54	0	133	0	126	0	156	1218	48	0	78	36970	108
	2. MaizeRouter	0	62	0	57	0	138	0	128	2	177	1348	51	0	80	32588	115
	3. BoxRouter	0	59	0	56	0	141	0	129	0	164	400	51	0	80	38976	112
	4. FastRoute	122	90	500	82	0	203	0	171	9680	252	1934	74	0	115	34236	155
	5. NTHU-R(3)	3474	79	3588	66	64	176	0	142	20630	258	5526	56	0	88	38146	161
	6. Bockenem	608	80	880	95	3266	178	396	157	3496	232	2754	84	0	99	100078	130
	7. NCTU-R(10)	3800	81	5178	76	98	184	8	160	16400	236	6722	68	0	105	34310	147
	8. FlexRouter	8698	65	7370	59	950	155	18	135	21802	181	7636	51	0	82	39488	119
	9. NTU2-R(13)	32520	62	13860	62	43332	402	4064	143	119822	438	6570	53	0	89	64130	119
	10. NTU1-R(9)	93608	58	24738	57	31178	142	1342	133	208804	166	17872	50	0	81	148646	117

TABLE V
COMPLETE SOLUTION STATISTICS FOR ALL ENTRIES TO THE ISPD 2007 GLOBAL ROUTING COMPETITION

VIII. ACKNOWLEDGEMENTS

We wish to thank Prof. Igor L. Markov from 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," *TCAD*, vol. 20, no. 5, pp. 622–632, 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. of ASP-DAC 2007*, 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," *TCAD*, vol. 15, no. 2, pp. 208–216, 1996.

[4] M. Cho and D. Z. Pan, "BoxRouter: A new global bouter based on box expansion and progressive ILP," in *Proc. of DAC 2006*, pp. 373–378.

[5] C. Chu, "FLUTE: fast lookup table based wirelength estimation technique," in *Proc. of ICCAD 2004*, pp. 696–701.

[6] C. C. N. Chu and Y.-C. Wong, "Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design," in *Proc. of ISPD 2005*, pp. 28–35.

[7] R. Goering, "IC routing contest boosts CAD research," http://www.eetimes.com/showArticle.jhtml?articleID=198500084, 2007.

[8] R. T. Hadsell and P. H. Madden, "Improved global routing through congestion estimation," in *Proc. of DAC 2003*, pp. 28–31.

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

[10] J. Hu and S. S. Sapatnekar, "A survey on multi-net global routing for integrated circuits," *Integration, the VLSI Journal*, vol. 31, no. 1, pp. 1–49, 2001.

[11] A. Kahng, I. Mandoiu, and A. Zelikovsky, "Highly scalable algorithms for rectilinear and octilinear steiner trees," in *Proc. of ASP-DAC 2003*, pp. 827–833.

[12] A. B. Kahng and X. Xu, "Accurate pseudo-constructive wirelength and congestion estimation," in *Proc. of SLIP 2003*, pp. 61–68.

[13] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Predictable routing," in *Proc. of ICCAD 2000*, pp. 110–113.

[14] ——, "Pattern routing: use and theory for increasing predictability and avoiding coupling," *TCAD*, vol. 21, no. 7, pp. 777–790, 2002.

[15] Z. Li, C. J. Alpert, S. T. Quay, S. S. Sapatnekar, and W. Shi, "Probabilistic congestion prediction with partial blockages," in *Proc. of ISQED 2007*, pp. 841–846.

[16] 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. of ISPD 2007*, pp. 127–134.

[17] J. Lou, S. Krishnamoorthy, and H. S. Sheng, "Estimating routing congestion using probabilistic analysis," in *Proc. of ISPD 2001*, pp. 112–117.

[18] M. Pan and C. Chu, "FastRoute: A step to integrate global routing into placement," in *Proc. of ICCAD 2006*, pp. 464–471.

[19] ——, "FastRoute 2.0: A high-quality and efficient global router," in *Proc. of ASP-DAC 2007*, pp. 250–255.

[20] ——, "IPR: An integrated placement and routing algorithm," in *Proc. of DAC 2007*, pp. 59–62.

[21] J. Roy and I. L. Markov, Personal communication, 2007.

[22] C.-W. Sham and E. F. Y. Young, "Congestion prediction in early stages," in *Proc. of SLIP 2005*, pp. 91–98.

[23] J. Westra, C. Bartels, and P. Groeneveld, "Probabilistic congestion prediction," in *Proc. of ISPD 2004*, pp. 204–209.

[24] J. Westra and P. Groeneveld, "Is probabilistic congestion estimation worthwhile?" in *Proc. of SLIP 2005*, pp. 99–106.

[25] http://www.ispd.cc/ispd07_contest.html

[26] http://www.ece.ucsb.edu/~kastner/labyrinth/

[27] http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/FastSteiner/

[28] http://home.eng.iastate.edu/~cnchu/flute.html

[29] http://www.cerc.utexas.edu/~thyeros/boxrouter/boxrouter.htm

[30] http://www.eecs.umich.edu/~mmoffitt/MaizeRouter/