To appear in *IEEE Transactions on CAD* 2008

High-performance Routing at the Nanometer Scale

Jarrod A. Roy and Igor L. Markov
The University of Michigan, Department of EECS
2260 Hayward Ave., Ann Arbor, MI 48109-2121
{royj,imarkov}@eecs.umich.edu

Abstract— **In this work we describe significant improvements to core routing technologies and outperform the best results from the ISPD '07 Global Routing Contest and ICCAD 2007 in terms of route completion and total wirelength.**

I. Introduction

Despite being one of the first areas of EDA to be automated in the 1960s, VLSI routing remains an area of active research and development as evidenced by a growing body of literature [2], [6], [15], [32], [33], recent collaboration between Cadence and IBM on routing technology [27], as well as the ISPD '07 Global Routing Contest organized by IBM Austin Research Laboratory [18]. Current efforts in routing are motivated by challenges present at the nanometer scale including: (i) very large wiring databases that require lean data structures and extremely efficient algorithms, (ii) sophisticated design rules that must be abstracted away during initial routing passes, (iii) relatively unreliable vias whose resistance may vary by up to 30 times [37], which requires via doubling [23], [25] and motivates additional effort to minimize via counts, (iv) signal integrity constraints and the dramatic impact of lateral capacitance on interconnect delay, which lead to wire density constraints, and (v) considerations of chemical mechanical polishing (CMP) that also lead to density constraints [8].

The ISPD '07 routing contest challenged the research community by distributing 16 very large routing benchmarks derived from recent chip layouts. Thanks to the wide participation in the contest and the public availability of the results, we observed an important trend which is illustrated in Figure 1 — routers that achieve low wirelength often suffer high violation counts, and routers that minimize violations often produce high wirelengths. Therefore, a key focus of our work is on adequate pricing of routing resources to balance interconnect length and congestion in multi-million gate designs, in a way that also allows to trade-off other nanoscale objectives and constraints. Additionally, the effective handling of vias, multiple metal layers and other aspects of nanoscale routing pose a series of algorithmic, implementation, benchmarking and integration challenges.

In this work we develop a high-performance routing technique based on Discrete Lagrange Multipliers (DLM), while pointing out inaccuracies, limitations and pitfalls of the related technique known as negotiated-congestion routing [28]. In particular, DLM offers a natural way to handle net weights and timing optimization in routing, and explains several empirical effects observed in negotiated-congestion techniques such as the last-gasp problem and the relative simplicity of 2-d formulations compared to multi-layer (3-d) formulations. Proposed algorithms are implemented in FGR[1], a high-performance global router for nanometer scale designs.

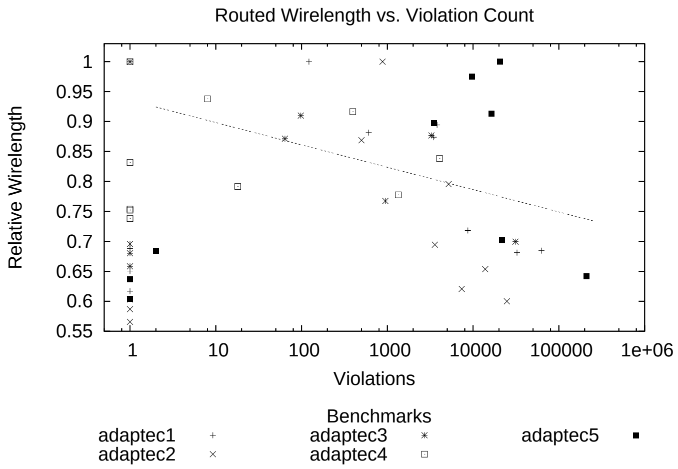


Fig. 1. Routed wirelength vs. violations for all competing routers on 2-d ISPD '07 benchmarks [18]. Note that violation counts are shown on a log-scale where 0 cannot be plotted, so completely legal solutions are depicted with exactly 1 violation. Relatively few solutions submitted to the contest were legal (35%), but they are generally a cut above the rest. Of the illegal solutions, as violations increase, routed wirelength decreases. To emphasize the trend, a linear least-squares fi t of the data has been added for the illegal solutions.

Our key contributions are:

- A routing technique based on Discrete Lagrange Multipliers (DLM) which provides a natural way to handle net weights and timing optimization in global routing. FGR handles two- and three-dimensional routing of ASICs with up to 870,000 nets.[2]
- Extensions of A*-search to restructure net topologies so as to avoid congestion and circumvent obstacles.
- Improved wirelength on the ISPD '07 Global Routing Contest suite [18]. FGR produces smaller wirelengths than the winners of the contest *on every benchmark*, and

[1]"Fairly Good Router"
[2]This is almost an order of magnitude greater than what has been reported in the literature for most ASIC and FPGA routers. In the 32-bit address space, FGR scales up to 1,000,000 nets, but can also be compiled to run in the 64-bit address space.

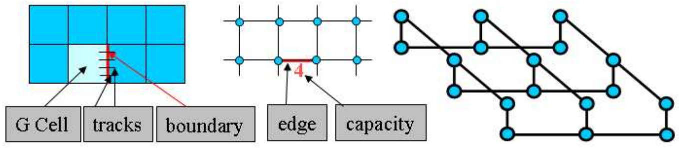


Fig. 2. Pictorial representations of the global routing grid. The images on the left and in the middle show how the layout is abstracted into a regular grid of GCells. GCells are represented by vertices, with adjacent vertices connected by graph edges. Capacities on edges that join GCells can be defined as the number of routing tracks that cross GCell boundaries. The image at the right shows that horizontal and vertical connections on different layers with vias connecting them.

is able to route without overflows every benchmark that the winners routed without overflows. In terms of wirelength, FGR outperforms BoxRouter 1.0 [6] by 10.6% and MaizeRouter [29] by 9.1%. We also achieve shorter wirelength than Archer [31] and BoxRouter 2.0 [7] *on every benchmark*, and obtain best violation counts on newblue1 of 234 – 41.4% better than BoxRouter 2.0.

- Violation-free routing of all ISPD '98 IBM benchmarks, unlike routers in previous literature. FGR uses 35% less runtime than BoxRouter 1.0 and produces solutions with 2.7% smaller wirelength.
- Thorough empirical evaluation of several routing strategies and algorithms including net decomposition by MST vs. Steiner trees and layer assignment for 3-d routing problems vs. direct 3-d maze routing. We identify previously unreported bottlenecks, such as the "last gasp" problem in negotiated-congestion routing, and propose solutions.

This paper is organized as follows. Section II, reviews relevant background and previous work. Section III describes the architecture of the FGR router, the mathematical basis for its key algorithms, and important insights into the integration of major components. We benchmark FGR against state of the art in Section IV and conclude in Section V.

II. Background and Previous Work

Routing plays a key role in VLSI physical design as it determines the specific shape and layout of interconnect, impacting performance, power and manufacturability. Routing is traditionally divided into the two steps of global and detail routing.

Global and Detail Routing. During global routing, complex design rules are abstracted away and a design is divided into a regular grid (see Figure 2). Routes are created for each net that connect adjacent grid cells. Capacities are assigned to pairs of adjacent grid cells to model limited routing resources between the cells. Since different metal layers may use distinct wire pitches, routing capacities at each layer may differ to reflect this. A global routing solution is legal if all nets are connected and all capacity constraints are satisfied.

Detail routing takes a global routing solution with a small number of capacity violations (overflows), or none at all, and assigns wires to routing tracks while enforcing spacing constraints and more sophisticated design rules. Starting with slightly illegal global routes can make detail routing considerably more difficult, therefore a global router must minimize violations and wirelength, seeking to avoid violations entirely when possible.

Traditional algorithms for detail routing often assume a specific, small number of metal layers and operate in isolated layout regions — channels or switch-boxes. However, over-the-cell routing with six or more metal layers made many such algorithms obsolete and lead to the adoption of similar graph-theoretical techniques in global and detail routing, perhaps with different layout, resource and delay models.

In our experience with Cadence WarpRoute, three quarters of total runtime is spent in detail routing, but the quality of global routes profoundly affects the runtime and success of detail routing. A recent proposal [32] suggests invoking a fast global router during global and detail placement, so as to mitigate wiring congestion early. This application is particularly attractive for sub-130nm technology nodes where lateral capacitance of wires is a major contributor to interconnect delay. In this context, accurate timing analysis requires information about regions through which a given net passes as well as wire density in these regions [41].

Maze Routing connects pairs of terminals on the routing grid using standard search techniques such as BFS and Dijkstra's algorithm [12]. More than 50% of nets in modern designs connect only two pins. BFS can find the shortest path between a source location and a target location, if one exists, but cannot handle routing segments with non-trivial weights. Dijkstra's algorithm can handle non-negative costs of routing segments, but is at least several times slower than BFS. A*-search is a minor modification to Dijkstra's algorithm that significantly improves speed during 2-d and 3-d routing [16]. In A*-search, a lower bound of the distance to the target is added to node priority in Dijkstra's algorithm. Straight-line distance is commonly used as a lower bound.

Pattern Routing [21] is a technique that severely restricts the number of ways in which a net can be routed to simplify the routing process. For example, L-shape routing seeks to implement each two-pin net with a single bend. This technique is surprisingly useful in ASIC routing and justified by via minimization. Empirical studies [43] show that in a fully-routed design a majority of all 2-pin nets take on L-shapes. In global routing, where minor detours are abstracted away, L-shapes are even more prevalent. Two-bend routes are often called Z-shapes, but generic pattern-based routing can consider any finite number of routing topologies for each net, and selects one of them. It is particularly amenable to Integer Linear Programming formulations [6], as described later in the section.

Multi-pin nets. Most global routing algorithms decompose nets with three or more pins into two-pin subnets at the beginning of global routing as this eases maze routing. This decomposition has been traditionally done using Minimal Spanning Tree (MST) algorithms, but using fast and extremely accurate Rectilinear Steiner Minimal Tree (RSMT) construction algorithms has become increasingly popular in the literature [6], [32], [33]. Four decompositions of a 5-pin net by Steiner trees and MSTs are shown in Figure 3.

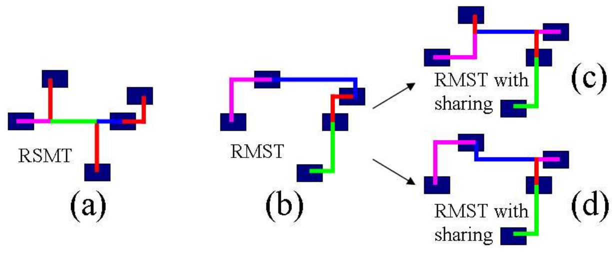


Fig. 3. Decomposition of a 5-pin net by minimal Steiner tree (a), MST (b) and MSTs with sharing (c)&(d). The choice of (c) or (d) depends on congestion. The minimal Steiner tree (a) contains 5 flat subnets and 1 L-shaped subnet, whereas the shared MST (d) has 2 flat subnets and 3 L-shaped subnets which gives it greater flexibility.

The RSMT tool FLUTE [9] is used by BoxRouter 1.0 [6], BoxRouter 2.0 [7] and FastRoute [32], [33]. FLUTE uses look-up tables for nets with nine or fewer pins and quickly builds optimal trees for such nets [9]. For larger nets, a divide-and-conquer method is employed [9]. FastSteiner [19] is another RSMT algorithm that is more scalable than most RSMT algorithms. FastSteiner does not guarantee optimality, but frequently produces solutions with smaller total wirelength than FLUTE for nets with more than nine pins. In Section IV-C we compare MST with sharing to a combination of FLUTE and FastSteiner, and find that Steiner constructors lead to smaller routed length but greater via counts.

Rip-up-and-re-route (RRR) takes an initial, usually illegal, routing solution and iterates greedy one-net-at-a-time routing passes for nets that compete for routing resources, but may change the ordering each time in hope to better reconcile these nets. In each iteration, nets that pass through congested regions are “ripped up” (all resources for the net are removed from the routing grid) and are rerouted with a maze router to use lesser congested regions. Major differences between various implementations [6], [13], [15], [28], [32], [33] include which nets are ripped up and rerouted at each iteration, the order in which to rip up nets and reroute them, if nets are allowed to be rerouted through areas that are already congested, and the costs associated with routing through a particular routing edge given its current congestion.

Congestion Amplification [15] was recently introduced as an improvement to pricing of routing resources during RRR. Many routers that employ RRR do not penalize nets for going through uncongested regions, and then drastically increase cost once a routing edge is full. The authors of [15] propose to use a more gradual linear cost function for edges before they become full in order to spread wires from areas that are likely to become congested. In addition, when congestion estimates are calculated after each iteration of RRR, regions with high congestion have their estimates artificially increased (amplified) and regions with low congestion have their estimates decreased. This provides a greater incentive for maze routers to avoid highly congested regions, often at the cost of increased wirelength.

Negotiated-congestion Routing (NCR) [28] was introduced in the mid-1990s for global routing in FPGAs and is used in VPR (the dominant place-and-route tool for FPGAs) [3], but has not seen much use in the ASIC literature. NCR builds upon RRR by gradually making routing edges that are consistently congested more expensive, encouraging the maze router to choose alternative routes when they are available. The cost c_e of routing edge e

$$c_e = (b_e + h_e) \cdot p_e \tag{1}$$

is a function of the base cost (b_e), added cost reflecting congestion history (h_e), and penalty for current congestion (p_e) [28]. NCR seeks to minimize $\sum_e c_e$.

To begin negotiated-congestion routing, each net is routed using the smallest possible wirelength regardless of edge capacities. Next, rip-up-and-re-route proceeds. At the beginning of a RRR iteration, the historical cost h_e of all over-capacity routing edges is increased:

$$h_e^{k+1} = \begin{cases} h_e^k + h_{inc} & \text{if } e \text{ is overfull} \\ h_e^k & \text{otherwise} \end{cases} \tag{2}$$

where h_{inc} is a constant. The choice of h_{inc} affects convergence time and solution quality: higher values lead to faster convergence but higher routed wirelength. After cost adjustment, each net of the design is individually ripped up and rerouted by a maze router. The authors suggest that only nets passing through congested regions need to be rerouted and this approach is used in FGR. The ordering of nets during rip-up-and-re-route is the same for each iteration, but can be chosen arbitrarily, according to the authors of [28], because the gradual cost increase in congested areas removes ties that require sophisticated net ordering techniques in traditional RRR implementations.

Reported implementations of NCR do not handle multi-layer routing and via minimization — key aspects of nanoscale ASIC layout. Additionally, NCR has not been validated in the literature at the scale of large ASIC netlists.

Multi-level routing techniques work similarly to those in partitioning [20] and placement [5]. The original routing problem is effectively made simpler through a series of coarsening stages where routing grid cells are combined and many nets become subsumed within a single cell. This adds a hierarchy to the routing formulation. At the top of the hierarchy is the coarsest form of the routing problem which is small enough to be solved with sophisticated techniques that may not scale to large routing instances such as multi-commodity flow based techniques [2], [16], described below. Essential to the coarsening stage is the proper aggregation of routing resources so that routing solutions at higher levels closely resemble valid routing solutions at lower levels.

After the coarsest level of the hierarchy has been routed, iterative refinement of the current routing solution begins. The problem is un-coarsened by one level and the current solution is adapted to the finer routing grid. This stage is nontrivial as nets can gain additional pins as the routing grid is refined and new nets that were previously subsumed by routing cells will become visible and need to be routed from scratch. This refinement process iterates until the finest level of the hierarchy, the original routing problem, has been successfully routed. Multi-level routers in the literature generally have smaller runtimes than flat techniques and show higher completion rates [10], [11].

Combinatorial optimization techniques. Other sophisticated techniques for routing have been proposed, such as the use of multi-commodity flows (MCF) [2], [16] and integer linear programming (ILP) [6]. Both of these techniques attempt to route nets simultaneously in order to avoid the problems associated with net ordering.

There are a variety of ILP formulations for routing. Many require that multi-pin nets be divided into a small number of two-pin topologies. For each two-pin net, one sets up several constraints to ensure that the pins are connected. Constraints are also added so that the number of nets passing through each routing segment does not exceed the capacity. Solving this formulation for all nets simultaneously will optimally solve the given routing problem if possible, but has its drawbacks including difficulty expressing non-linear delay models.

Multi-commodity flow (MCF) techniques take a different approach to solve the ILP formulation by relaxing it into a linear programming (LP) formulation. An approximation algorithm which successively adjusts routing edge weights and builds new weighted Steiner trees per net at each iteration is used to solve the LP. ILP-based BoxRouter 1.0 has been compared to a recent MCF-based router [6] and found to be superior in speed and solution quality. Additionally, MCF techniques offer less flexibility in terms of objective functions and constraints than the RRR and NCR frameworks.

FastRoute [32], [33] uses a simplified, more greedy form of RRR and finishes orders of magnitude faster than other routers. However, it was able to legally route only 6 of 16 benchmarks at the ISPD '07 contest [18], while other routers completed up to 12 benchmarks without violations. Additionally, on the easier ISPD '98 benchmarks, it routes fewer benchmarks than FGR (see Table III).

FastRoute 1.0 [32] first uses FLUTE to decompose nets and estimate congestion in the design, then attempts to restructure Steiner trees to avoid congestion. FastRoute 2.0 [33] features the following modification of RRR. When a single subnet is ripped up, the net to which the subnet belongs will be separated into two connected components. It becomes the maze router's job to connect the two components of the net in the least costly way. While this optimization allows the router to move Steiner points away from congested regions, it invalidates the point-to-point lower bound on which A*-search relies. Hence, the slower Dijkstra's algorithm must be used instead.

BoxRouter 1.0 [6] avoids fine-grain net ordering in congested regions through the use ILP formulations. BoxRouter 1.0 decomposes nets using Steiner trees produced with FLUTE but never re-examines their decomposition. Next it performs a pass of pattern routing that identifies the most congested rectangular region, where it formulates an ILP to route as many nets using L-shapes as possible. Remaining nets are routed by the maze router, using as few resources outside the region as possible. Next, the region is expanded, and an incremental ILP formulation is used. This cycle repeats until the entire layout is covered by the expanding region.

Multilevel Advanced Routing System (MARS) [11] is a multi-level router based on the techniques first presented in [10] with several important enhancements. The first is that MARS performs accurate resource reservation during the coarsening phase of multi-level routing. This takes into account those nets which are subsumed into the coarsened routing grid and removes resources for them. This results in more accurate resource counts at higher levels of the routing hierarchy which better represent the original routing problem. The second enhancement is that MARS divides multi-pin nets using congestion-driven Steiner trees. At each level of the routing hierarchy, each net is examined and new Steiner trees are built to divide multi-pin nets. First MSTs are built for each net using the routing grid and not purely based on HPWL. Next, the edges of the MST for a particular net are sorted based on length and maze search is performed to join the edge to any other part of the existing tree. The new attachment points become Steiner points, and the Steiner tree for the net is formed from all of the paths found during maze search. Lastly, MARS uses historical costs based on congestion, calculated differently than in NCR, to price routing edges during maze routing.

III. High-performance Global Routing

In this section we describe the architecture of FGR, the mathematical basis for its key algorithms, and important implementation insights.

A. Basic Algorithmic Framework

Routing algorithms must carefully balance wirelength minimization and congestion. Some detours may be necessary to avoid routing violations and overcapacity GCells, but excessive detouring leads to overconsumption of routing resources, aggravating congestion. In particular, the results of the ISPD '07 routing contest [18] show that some routers are good at finding violation-free solutions, some are good at minimizing wirelength, but few are good at both. This trend is illustrated in Figure 1 which shows routed wirelength vs. violation count for 2-d solutions submitted to the contest. A likely source of this inflexibility is the common use of uniform, predetermined rules in all regions of the chip as in FastRoute [32], [33] and the Chi dispersion router [15].

In continuous optimization, dynamic pricing of constraint satisfaction can be modeled by Lagrange multipliers — a mathematical method for optimizing a multivariate function subject to a number of constraints [22]:

$$\begin{array}{c} \min_{\mathbf{x}\in X} W(\mathbf{x}) \\ \text{subject to}\;\; C_e(\mathbf{x}) = 0, \quad 1 \le e \le n \end{array} \tag{3}$$

The constrained optimization is reduced to the unconstrained optimization of the Lagrangian function F

$$F(\mathbf{x}, \lambda) = W(\mathbf{x}) + \sum_{e=1}^{n} \lambda_e C_e(\mathbf{x}) \tag{4}$$

where $\lambda = (\lambda_1, \ldots, \lambda_n)$ are positive real-valued Lagrange multipliers. In the case of routing, $C_e(\mathbf{x})$ represents the overflow penalty of routing edge e. $W(\mathbf{x})$ represents the total wirelength of routing solution $\mathbf{x}$ and is usually defined as a sum over nets or routing edges

$$W(\mathbf{x}) = \sum_{i=1}^{m} R_i(\mathbf{x}) = \sum_{e=1}^{n} B_e(\mathbf{x}) = \sum_{e=1}^{n} \left(\sum_{\text{net } i \text{ uses } e} b_e \right) \quad (5)$$

where $R_i(\mathbf{x})$ is the number of segments used by net i and $B_e(\mathbf{x})$ is the number of nets passing through edge e. Thus Equation 4 can be rewritten

$$F(\mathbf{x}, \lambda) = \sum_{e=1}^{n} \left(B_e(\mathbf{x}) + \lambda_e C_e(\mathbf{x}) \right) \quad (6)$$

Here both original unknowns **x** and the Lagrange multipliers $\{\lambda_e\}$ are considered variables subject to optimization. For large, sparse, convex problems, iterative techniques are used, such as *steepest descent*, *Newton's method*, etc. In particular, Lagrange multipliers are updated additively as follows

$$\lambda^{k+1} = \lambda^k + \alpha C(\mathbf{x}^k) \quad (7)$$

where $\alpha > 0$ is a line-search parameter. Note the similarity in the update of the Lagrange multipliers and how h_e is updated in Equation 2. While routing instances are large, sparse problems, they are discrete and non-convex. This calls for a different iterative optimization procedure, such as *greedy search*, *hill-climbing* or *rip-up-and-re-route*. However, since Lagrange multipliers remain continuous, the same update rule can be adopted.

Interpreting Equation 6 for a single edge e using the notation of NCR, c_e is derived as

$$c_e = b_e + h_e \cdot p_e \quad (8)$$

which is different than Equation 1 [28], but also is more intuitive since it preserves the base cost. Therefore FGR uses this Discrete Lagrange Multiplier (DLM) formulation instead of NCR which was used in FGR's ISPD '07 contest submission. To compute p_e, FGR uses a new penalty function introduced in Section III-B below. Furthermore, the justification of dynamic cost updates through DLMs explains the results shown in Sections III-E, III-F and IV.

While Lagrangian relaxation has been suggested for global routing before, all uses we are aware of are either (1) specific to timing-driven routing and maintain net-centric Lagrange multipliers [24], [30] or (2) focus on a single net at a time [31]. These algorithms use conventional history-based rip-up and re-route for the router's main loop.[3]

In addition to being a rigorous mathematical technique, the use of Lagrange multipliers often admits application-specific interpretation. For example, it is used in macro-economics to mathematically describe market pricing — in a market economy, adequate resource pricing encourages consumers to look for competitive alternatives, leaving the most expensive resources to the consumers that gain most. A very similar interpretation holds in the case of routing, and the "fairness" of this pricing system is confirmed by good convergence properties in practice, as illustrated in Figure 8.

[3]The authors of [24] briefly mention the similarity of history-based rip-up and re-route to Lagrangian relaxation, but make no modifi cations to the NCR formulation.

In the initial routing formulation (Equation 3) all nets are treated equally when optimizing total wirelength, but in many cases certain nets are more important than others for optimization, as in timing-driven routing. Each net is assigned a weight, and the goal is to optimize total weighted wirelength. Weighted wirelength is written as

$$W'(\mathbf{x}) = \sum_{i=1}^{m} w_i R_i(\mathbf{x}) = \sum_{e=1}^{n} B'_e(\mathbf{x}) \quad (9)$$

where w_i is the weight of net i and $B'_e(\mathbf{x})$ is the total weight of nets passing through routing edge e

$$B'_e(\mathbf{x}) = \sum_{\text{net } i \text{ uses } e} w_i \cdot b_e \quad (10)$$

By replacing B_e in Equation 6 with B'_e, the Lagrange relaxation becomes

$$F(\mathbf{x}, \lambda) = \sum_{e=1}^{n} \left(\left(\sum_{\text{net } i \text{ uses } e} w_i \cdot b_e \right) + \lambda_e C_e(\mathbf{x}) \right) \quad (11)$$

As a result, the cost c_e of edge e during maze routing is different for different nets that may be routed through it and must be rewritten as $c_e(i)$

$$c_e(i) = w_i \cdot b_e + h_e \cdot p_e \quad (12)$$

Note that the original NCR formulation does not separate b_e and makes it difficult to account for net weights.

To gauge the effectiveness of net weighting in DLM, we choose a random subset of 10% of the nets of the ISPD '07 benchmark newblue2 and increase their weight from the default of 1 to 2 and route from scratch. Distributions of detours on the nets are shown in Figure 4. Detouring on the nets with higher weight is reduced as is the overall detouring on the design. Runtime and total wirelength are affected negligibly. Thus using net weights is an effective method for controlling detouring and on selected nets.

B. Congestion Penalty

Let r_e and u_e represent the resources and current usage of a routing edge e and define the relative overflow $\omega_e = u_e/r_e$. The congestion penalty term p_e for edge e is computed as a function of ω_e.

$$p_e = \begin{cases} \exp\left(k(\omega_e - 1)\right) & \text{if } \omega_e > 1 \\ \omega_e & \text{otherwise} \end{cases} \quad (13)$$

The exponential nature of our cost function for overfull routing edges serves to amplify congestion and gives the maze router incentive to avoid overfull edges when re-routing nets (see Figure 5, where $k = \ln 5$). We have studied $0 < k \leq \ln 10$ and found that higher values of k reduce runtime, but increase detouring and routed length. FGR uses $k = \ln 5$ by default. Instead of using uniform weights of 1 for routing edges to create an initial routing solution, which is common in NCR, FGR uses $b_e + p_e$ as the weight for edges to create an initial solution, where p_e is calculated on the fly per routing edge according to Equation 13.

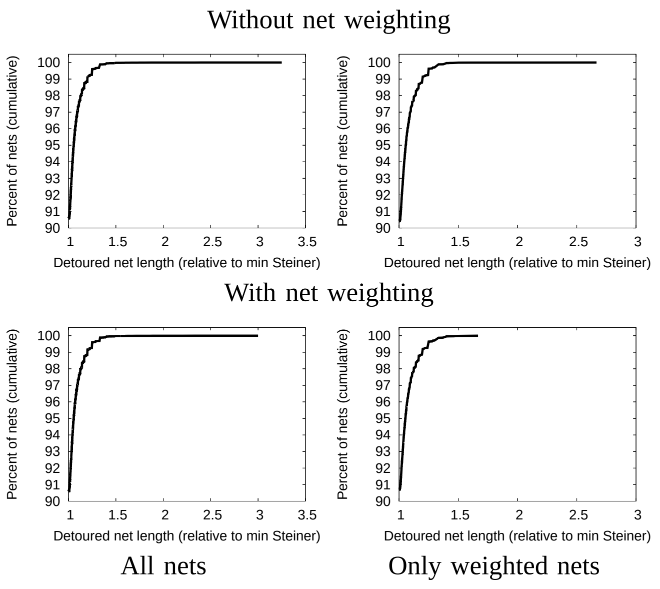


Fig. 4. Cumulative distributions of detouring without (above) and with (below) net weighting on the 2-d newblue2 benchmark. Net detours are measured as a ratio of routed net length to Steiner wirelength as given by FLUTE [9]. When weights are applied to a subset of the nets, detouring on those nets is reduced significantly without increasing detouring of other nets.

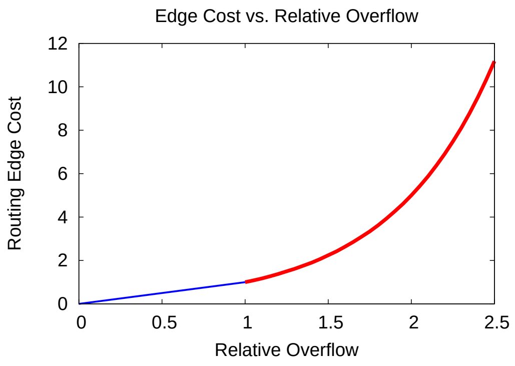


Fig. 5. Cost of a routing edge as a function of relative overflow. Cost is linear while the edge is not overfilled, but grows exponentially once the edge is overfull.

C. Via Pricing and Optimization

The closest previous works to ours — those on NCR — do not consider via minimization because they focus on FPGA routing. Nonetheless, given that the resistivity of tungsten (the material of vias) is much higher than that of copper and aluminum, vias are critical in timing-driven routing. An unnecessarily large number of vias can hamper routability because each via obstructs a section of its track. Furthermore, the high variability in via parasitics [37] and the common practice of post-route via doubling to improve yield [23], [25] suggest that via minimization is a key issue in routing at the nanometer scale. Table I illustrates just how significant vias are in the ISPD '07 contest benchmarks. Vias represent from 26% to 49% of the total cost of FGR's solutions to the 2-d benchmarks. Comparing two-layer routing with 6-layer routing, via counts approximately triple and account for 50% to 74% of total cost.

To model the cost of vias, FGR treats them as segments in the routing graph. These segments connect adjacent routing layers as shown in Figure 2 and have unlimited capacity. Via routing segments have a different base cost, usually higher than that for regular segments. This flexibility allows FGR to price vias in specific applications. For example, in the ISPD '07 contest one via is equivalent to three routing grid segments, so the cost of vias in FGR is set to $3b_e$.

TABLE I

Routed cost breakdown of FGR's solutions to the ISPD '07 Global Routing Contest benchmarks [18]. "FLUTE Ratio" is the ratio of the length of routing segments used to the Steiner tree length of all nets as computed by FLUTE [9]. Vias account for more than 25% of total cost in every 2-d benchmark and more than 50% of total cost in each 3-d benchmark, highlighting the importance of via minimization.

Benchmark	Segment WL (e5)	FLUTE ratio	Vias (e5)	Total cost (e5)	Via cost %
adaptec1 2-d	35.88	1.0594	6.19	54.44	34.09%
adaptec1 3-d	36.37	1.0739	17.36	88.45	58.88%
adaptec2 2-d	33.21	1.0371	6.36	52.30	36.50%
adaptec2 3-d	33.74	1.0536	18.72	89.89	62.47%
adaptec3 2-d	96.09	1.0295	11.60	130.89	26.59%
adaptec3 3-d	97.02	1.0395	34.21	199.66	51.41%
adaptec4 2-d	90.02	1.0143	11.66	125.00	27.98%
adaptec4 3-d	91.28	1.0285	30.56	182.96	50.11%
adaptec5 2-d	102.79	1.0499	16.45	152.13	32.43%
adaptec5 3-d	103.89	1.0612	52.03	259.98	60.04%
newblue1 2-d	24.15	1.0400	7.76	47.42	49.07%
newblue1 3-d	24.15	1.0400	23.37	94.26	74.38%
newblue2 2-d	46.81	1.0179	9.90	76.51	38.82%
newblue2 3-d	47.91	1.0418	28.08	132.16	63.75%
newblue3 2-d	75.63	1.0253	11.20	109.23	30.76%
newblue3 3-d	75.63	1.0253	32.69	173.71	56.46%

Assigning via segments non-zero costs in the routing grid allows A*-search to naturally optimize via counts when finding shortest paths. However, to use A*-search, an accurate lower bound for path cost is also needed. One could ignore vias in the lower bound calculation, but FGR uses the layer difference of the source and target which is more accurate.

D. Interactions Between Single- and Multi-Net Routing

FGR initially decomposes nets using an RSMT or RMST topology. However, given that congestion-driven Steiner trees are not easy to construct and precise congestion in every GCell is not known beforehand, it is important to modify net topologies during multi-net routing.

Figure 6 compares the net decomposition and restructuring techniques used by FGR to those in prior work. During DLM, the most congested subnets are ripped up and rerouted by A*-search. When ripping up a subnet with endpoints P_1 and P_2, FastRoute 2.0 tries to reconnect the two components of the net, not necessarily using P_1 or P_2, which requires a more sophisticated lower bound than Manhattan distance to use A*-search. When re-routing a subnet, FGR requires the replacement segments to pass between P_1 and P_2, based on the following result.

Theorem 1: Consider shortest paths between two trees embedded into the routing grid. Let P_1 and P_2 be nodes arbitrarily selected in the trees T_1 and T_2, respectively. If the costs of routing edges taken by tree segments are set to zero, then there is a one-to-one correspondence between (i) shortest paths

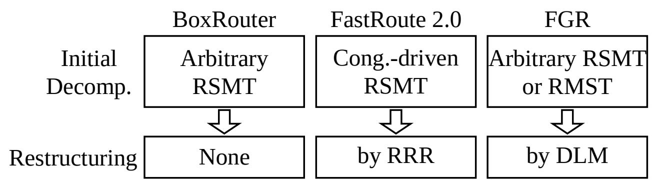


Fig. 6. A comparison of the net decomposition techniques used by BoxRouter 1.0 [6], FastRoute 2.0 [33] and FGR. In Section IV-C, we compare the use of RMSTs and RSMT in FGR.

between T_1 and T_2 and (ii) shortest paths between P_1 and P_2.

Proof: Assume $\exists$ a shortest path $A \to B$ joining T_1 and T_2 such that $A \in T_1$ and $B \in T_2$. $\exists$ unique non-self-intersecting paths $P_1 \to A$ and $B \to P_2$ consisting of edges contained in T_1 and T_2, respectively. $cost(P_1 \to A) = cost(B \to P_2) = 0$. Thus $cost(P_1 \to A \to B \to P_2) = cost(A \to B)$. For the sake of contradiction, assume $P_1 \to A \to B \to P_2$ is not a shortest path; $\exists$[4] path $P_1 \rightsquigarrow P_2$ with $cost(P_1 \rightsquigarrow P_2) < cost(A \to B)$. $P_1 \rightsquigarrow P_2$ connects T_1 and T_2, so $cost(P_1 \rightsquigarrow P_2) \geq cost(A \to B)$. Contradiction.
Conversely, let $P_1 \to P_2$ be a shortest path. Let C be the last vertex along $P_1 \to P_2$ such that $C \in T_1$ and let D be the first vertex along $P_1 \to P_2$ such that $D \in T_2$. $\exists$ unique non-self-intersecting paths $P_1 \to C$ and $D \to P_2$ consisting of edges contained in T_1 and T_2, respectively. $cost(P_1 \to C) = cost(D \to P_2) = 0 \Rightarrow cost(P_1 \to P_2) = cost(C \to D)$. Assume for the sake of contradiction $C \to D$ is not a shortest path; $\exists$ path $A \rightsquigarrow B$, $A \in T_1, B \in T_2$, with $cost(A \rightsquigarrow B) < cost(C \to D) = cost(P_1 \to P_2)$. $\exists$ $P_1 \rightsquigarrow A$ and $B \rightsquigarrow P_2$ such that $cost(P_1 \rightsquigarrow A) = cost(B \rightsquigarrow P_2) = 0 \Rightarrow cost(P_1 \rightsquigarrow A \rightsquigarrow B \rightsquigarrow P_2) = cost(A \rightsquigarrow B) < cost(P_1 \to P_2)$. Contradiction. $\square$

Temporary change of edge costs to 0 is easy to implement during A*-search because nets are routed individually and any cost adjustments can be reverted before considering other nets. However, in order to use A*-search, a correct lower bound must be supplied. FGR normally uses the 3-d Manhattan distance multiplied by the minimum cost of any routing segment. The naive solution — to ignore the 0-cost edges — may produce estimates that are greater than the true cost, which would cause A*-search to produce incorrect solutions. However, literally setting an edge's cost to zero forces the lower bound will to be zero. Therefore, in our implementation we set the cost of previously used edges to $\varepsilon > 0$, a small value. We call this technique ε-sharing and illustrate it in Figure 7, where FGR modifies the net topology to avoid congestion.

While prior state-of-the-art routers (BoxRouter, FastRoute and MaizeRouter) consistently start by decomposing multi-pin nets with minimal Steiner trees, we believe that integration of ε-sharing into a powerful DLM framework facilitates additional opportunities. As illustrated in Figure 3, Steiner trees tend to generate net decompositions with many flat subnets which offer no flexibility in routing. MSTs tend to

[4]Here, $\rightsquigarrow$ denotes paths assumed to exist for the sake of contradiction.

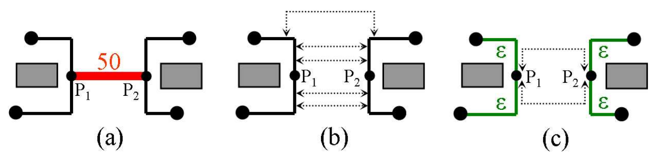


Fig. 7. Re-routing a subnet and changing net topology in FGR. The shaded boxes represent obstacles. The tree in (a) passes through a congested segment in the middle which must be ripped up. The dashed arrows in (b) represent several possible re-routings that a restructuring algorithm may consider. The re-routings shown in (c) are two that FGR will consider during DLM. Paths considered by FGR must start and end along the endpoints of the segment that was removed. Both of these re-routings reuse routing segments from the net and create new Steiner points if chosen. The use of temporary zero-cost edges is required to preserve the efficiency of A*-search.

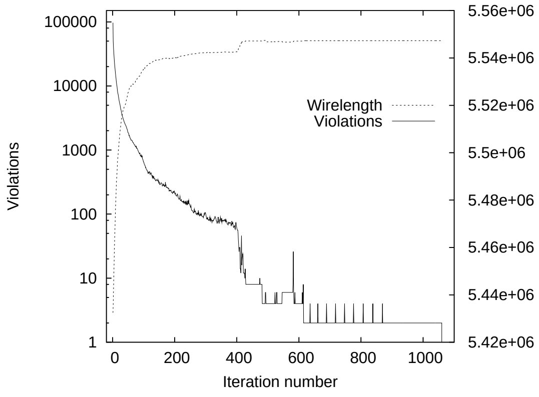
have fewer edges but with more flexibility, which can be exploited by DLM to avoid congestion. Moreover, the gradual addition of sharing to MSTs during DLM-based topology restructuring can generate high-quality congestion-driven Steiner trees without the need to estimate congestion before routing. Starting with minimal Steiner trees seems to require heavier restructuring to achieve similar effects, and could not only slow down maze routing, but also make RRR or DLM less successful. Using RSMTs vs. RMSTs is covered in Section IV-C.

E. *Overcoming the "Last Gasp" Problem*

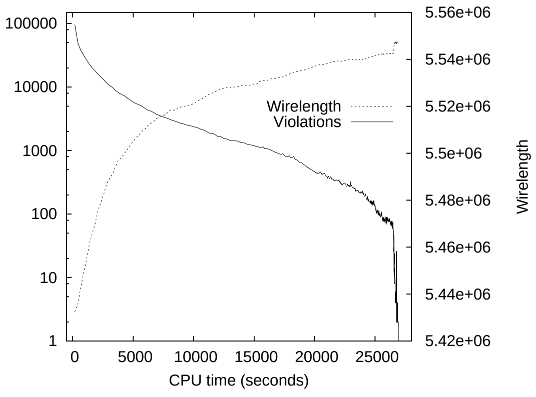
Discrete Lagrange multipliers work well at the large scale because the statistical behavior of numerous discrete variables is not very different from the continuous case. However, when only several violations remain, the routing task becomes much more discrete. In our experiments with almost every benchmark we have observed unusual behavior where FGR spends many DLM iterations when its solution is nearly legal before it is able to terminate with a completely legal solution. Indeed, more than 75% of DLM's iterations for the adaptec2 benchmark [18] take place when less than 0.01% of routing segments have overflow. We term this undesirable behavior the "last gasp" problem and illustrate it on the adaptec1 2-d benchmark in Figure 8. To rectify this situation, we propose the following improvement. When the percentage of routing edges with overflow becomes small, restrict the maze router to using only edges that have available space and weigh routing edges only by their base cost b_e. Thus if there is any way to route the net without causing overflow, it will be used to avoid further rip-up iterations. Otherwise, default DLM is used. In many cases this last phase of DLM reduces iterations without impacting total routed wirelength.

F. *3-dimensional Routing*

The difficulties experienced by DLM due to discreteness also suggest that traditional 2-d routing may be considerably easier than proper 3-d routing where smaller edge capacities are spread through multiple routing layers. In other words, aggregating edge capacities in one layer would encourage continuous-like resource pricing, making it easier to satisfy all constraints. This is consistent with what experimental observations discussed in Section IV-D.



(a)



(b)

Fig. 8. Violation count and wirelength on the 2-d ISPD ‘07 benchmark adaptec1 plotted as a function of (a) iteration number and (b) time. Violation counts are plotted on a log-scale and decrease, while wirelength is plotted on a linear scale and monotonically increases. Note that the majority of DLM iterations occur when 100 or fewer violations remain, but total wirelength noticeably increases during that phase.

```
Input: 2-d routing solution, 2dsol
Output: 3-d routing solution, 3dsol
1  foreach net n in 2dsol
2    foreach subnet s of n
3      route = 2dsol.getRoute(s)
4      currPoint = s.terminal1
5      currLayer = currPoint.layer
6      while(currPoint != s.terminal2)
7        nextPoint = route.getNextPoint(currPoint)
8        find nextLayer: the layer closest to
          currLayer where adding an edge connecting
          currPoint and nextPoint causes least overflow
9        add segment from currPoint to nextPoint
          on layer nextLayer to 3dsol
10       add vias connecting
          (currPoint.x,currPoint.y,currLayer) and
          (currPoint.x,currPoint.y,nextLayer) to 3dsol
11       currPoint = nextPoint
12       currLayer = nextLayer
13     add vias connecting
        (currPoint.x,currPoint.y,currLayer) and
        (currPoint.x,currPoint.y,s.terminal2.layer)
        to 3dsol
```

Fig. 9. Layer assignment in FGR.

FGR performs 3-d routing by first projecting the routing instance onto a 2-d grid and aggregating the capacities of edges that project onto each other. This grid contains a single layer of horizontal wires and a single layer of vertical wires connected by a layer of vias, such as the grid depicted at the right of Figure 2. Capacities on higher layers may be smaller due to increased pitch, but for each routing grid edge FGR calculates the number of wires that are allowed to pass through it, which takes wire widths and pitches into account. FGR routes this 2-d problem instance as normal until a legal solution is found or a runtime/iteration limit is reached. Next FGR performs layer assignment for each routing segment used in the 2-d solution.

Theorem 2: If the 2-d instance generated as a result of the aggregating process describe above has a legal solution and via counts are unconstrained, the original 3-d instance must have a legal solution.

Proof: 3-d routes can be constructed by the algorithm in Figure 9. □

FGR’s method will produce a 3-d solution that uses exactly the same number of routing segments as the 2-d solution, but differ in via counts. Unfortunately the difference in via counts is usually large and proportional to the number of layers in the 3-d instance. To counteract this phenomenon, FGR performs full 3-d cleanup which consists of a single round of RRR for every subnet to reduce vias. In this round of optimization, the cost of each routing segment is much simpler than in DLM: each routing segment is assigned a cost of 1 and vias are priced as in Section III-C. It is easy to lower-bound the cost of a path with these edge costs by the 3-d Manhattan distance, so it is particularly amenable to A*-search. Each subnet is ripped up and rerouted by the maze router individually, and edges with no spare capacity are not allowed. While Theorem 2 is not a surprising result, the fact that direct 3-d routing is less successful than 2-d routing with 3-d post-processing was unexpected and, in fact, undermined FGR’s performance in the ISPD ‘07 routing contest.

IV. Experimental Results

We have implemented FGR in C++ without external libraries (compiled with GCC 3.4.5), but added optional interface to the Steiner-tree packages FLUTE [9] and FastSteiner [19] to compare them with MST decompositions. The core algorithms and data structures of FGR were implemented in one month. All runs were performed on 2.4 GHz Opteron workstations running Linux. FGR was compiled in 32-bit mode and was therefore limited to less than 4GB of RAM.

A. Performance on ISPD ‘98 Benchmarks

Table II describes the ISPD ‘98 IBM benchmarks and compares FGR to BoxRouter 1.0 [6] in terms of runtime. Table III compares FGR to BoxRouter 1.0 and FastRoute 2.0 [33] in terms of solution quality. Unlike all previous routers in the literature, FGR is able to route all of the IBM designs without overflow. Both BoxRouter 1.0 and FastRoute 2.0, which report the best results on this suite so far, produce solutions with overflow on 4 and 3 of the benchmarks, respectively. Overall, FGR produces solutions with 2.7% less wirelength than BoxRouter 1.0 and 3.6% less wirelength than FastRoute 2.0. In addition, FGR is faster than BoxRouter 1.0 on 7 of the 10 benchmarks and uses 35% less runtime to complete the entire suite. Unlike

TABLE II

STATISTICS OF THE ISPD '98 IBM BENCHMARK SUITE [17]. RUNTIMES FOR BOXROUTER 1.0 [6] AND FGR 1.0 ARE GIVEN IN SECONDS. FGR IS FASTER THAN BOXROUTER ON 7 OF THE 10 BENCHMARKS AND USES 35% LESS RUNTIME TO SOLVE THE ENTIRE SUITE.

Bench-mark	# nets	Grid	Router runtime (s)	
			BoxRouter 1.0	FGR 1.0
ibm01	11507	64×64	6	10
ibm02	18429	80×64	25	13
ibm03	21621	80×64	13	5
ibm04	26163	96×64	18	29
ibm05	27777	128×64	37	6
ibm06	33354	128×64	25	18
ibm07	44394	192×64	39	20
ibm08	47944	192×64	68	18
ibm09	50393	256×64	50	20
ibm10	64227	256×64	73	92
Total			354	231

TABLE III

COMPARISON OF FGR 1.0 TO FASTROUTE 2.0 [33] AND BOXROUTER 1.0 [6] ON THE ISPD '98 IBM BENCHMARK SUITE [17]. FGR COMPLETES ALL 10 OF THE BENCHMARKS WHILE BOXROUTER 1.0 AND FASTROUTE 2.0 LEAVE OVERFLOW ON 4 AND 3 OF THE BENCHMARKS, RESPECTIVELY. IN TERMS OF ROUTED WIRELENGTH, FGR OUTPERFORMS BOXROUTER 1.0 BY 2.7% AND FASTROUTE 2.0 BY 3.6%.

Bench-mark	BoxRouter 1.0		FastRoute 2.0		FGR 1.0		vs. Box-Router 1.0	vs. Fast-Route 2.0
	ovfl	WL	ovfl	WL	ovfl	WL		
ibm01	**102**	65588	**31**	68489	0	**63332**	-3.44%	-7.53%
ibm02	**33**	178759	0	178868	0	**168918**	-5.51%	-5.56%
ibm03	0	151299	0	150393	0	**146412**	-3.23%	-2.65%
ibm04	**309**	173289	**64**	175037	0	**167101**	-3.57%	-4.53%
ibm05	0	409747	–	–	0	**409739**	-0.00%	–
ibm06	0	282325	0	284935	0	**277608**	-1.67%	-2.57%
ibm07	**53**	378876	0	375185	0	**366180**	-3.35%	-2.40%
ibm08	0	415025	0	411703	0	**404714**	-2.48%	-1.70%
ibm09	0	418615	**3**	424949	0	**413053**	-1.33%	-2.80%
ibm10	0	593186	0	595622	0	**578795**	-2.43%	-2.83%
Average							**-2.71%**	**-3.64%**

the ISPD '07 contest benchmarks, the ISPD '98 benchmarks feature only a single metal layer, making via minimization unnecessary.

B. Performance on ISPD '07 Benchmarks

Table IV shows statistics of the benchmarks used at the ISPD '07 Global Routing Contest [18]. These benchmarks are considerably larger than the ISPD '98 benchmarks and include both two- and three-dimensional variants. These benchmarks also feature non-trivial routing obstacles, and, consequently, routing resources are not spread evenly throughout the layout as in the ISPD '98 suite. Table IV also shows runtimes and memory requirements for FGR on these benchmarks. In all cases FGR stays within the 32-bit memory space and finishes well under a given 24-hour timeout on all but the newblue1 and newblue3 benchmarks on which no router at the ISPD '07 contest was able to find a legal solution.[5]

Next, we compare FGR to the routers that scored best at the ISPD '07 contest. Since an earlier version of FGR placed 1st in the 2-d category, we exclude it from comparison (however, the version we report improves upon FGR's results in the contest on every benchmark). In Table V, we compare best-seen results for FGR 1.1 to MaizeRouter [29] which placed 1st in 3-d and 2nd in 2-d, and to BoxRouter 1.0 which placed 2nd in 3-d and 3rd in 2-d. FGR produces smallest wirelengths *on every benchmark* and is able to route without overflow every benchmark that was legally routed at the contest. In particular, FGR outperforms BoxRouter 1.0 in wirelength by 10.6% and MaizeRouter by 9.1%. Comparing FGR 1.1 to most recent routers, FGR 1.1 outperforms Archer [31] in wirelength by 10.1% and BoxRouter 2.0 [7] by 4.9%.

[5]FGR can be stopped much earlier, with only a slight increase in overflows.

TABLE IV

STATISTICS OF THE ISPD '07 GLOBAL ROUTING CONTEST BENCHMARKS [18]. FOR FGR 1.0 WE LIST RUNTIME (IN MINUTES), THE NUMBER OF ITERATIONS OF RIP-UP-AND-RE-ROUTE (WHICH ARE VERY SIMILAR FOR 2-D AND 3-D VARIANTS), AND MAXIMUM MEMORY USAGE, WHICH IS SIGNIFICANTLY GREATER FOR 3-D THAN FOR 2-D VARIANTS.

Bench-mark	# nets	Grid	FGR 1.0 on 2-d variants		FGR 1.0 on 3-d variants	
			time (m)	rip-ups	time (m)	memory
adaptec1	219794	324×324	451	557	430	869 MB
adaptec2	260159	424×424	56	2930	64	960 MB
adaptec3	466295	774×779	179	284	243	2393 MB
adaptec4	515304	774×779	19	47	55	2377 MB
adaptec5	867441	465×468	713	790	740	2309 MB
newblue1	331663	399×399	1441	983	1442	1154 MB
newblue2	463213	557×463	4	20	10	1621 MB
newblue3	551667	973×1256	1555	23	1501	3676 MB

TABLE V

COMPARISON OF BEST RESULTS OF FGR 1.1 TO THE OTHER TOP-3 ROUTERS AT THE ISPD '07 GLOBAL ROUTING CONTEST [18]. FGR ROUTES AS MANY BENCHMARKS WITHOUT OVERFLOW AS THE WINNERS OF THE CONTEST WITH 8.1% BETTER WIRELENGTH THAN THE BEST OF BOXROUTER 1.0 [6] AND MAIZEROUTER [29] AND BEST OVERFLOW RESULTS ON THE NEWBLUE1 2-D AND 3-D BENCHMARKS.

Bench-mark		Best of BoxRouter and MaizeRouter				FGR 1.1 (best-seen)			
		Overflow		Cost	Router	Overflow		Cost	vs.
		total	max	(e5)		total	max	(e5)	Best
adaptec	#1 2-d	0	0	58.84	Box	0	0	**53.71**	-8.72%
	#1 3-d	0	0	99.61	Maize	0	0	**88.02**	-11.64%
	#2 2-d	0	0	55.69	Box	0	0	**51.86**	-6.88%
	#2 3-d	0	0	98.12	Maize	0	0	**89.96**	-8.32%
	#3 2-d	0	0	137.75	Maize	0	0	**130.30**	-5.41%
	#3 3-d	0	0	214.08	Maize	0	0	**200.14**	-6.51%
	#4 2-d	0	0	128.45	Maize	0	0	**123.97**	-3.49%
	#4 3-d	0	0	194.38	Maize	0	0	**178.90**	-7.96%
	#5 2-d	0	0	164.32	Box	0	0	**151.47**	-7.82%
	#5 3-d	0	0	298.08	Box	0	0	**260.53**	-12.60%
newblue	#1 2-d	400	2	51.13	Box	234	2	**46.42**	-9.21%
	#1 3-d	400	2	101.83	Box	238	2	**90.68**	-10.95%
	#2 2-d	0	0	79.64	Maize	0	0	**75.78**	-4.85%
	#2 3-d	0	0	139.66	Maize	0	0	**129.30**	-7.42%
	#3 2-d	32588	1236	114.63	Maize	38386	1196	**107.28**	-6.41%
	#3 3-d	32840	1058	184.40	Maize	38398	400	**163.41**	-11.38%
Average									**-8.13%**

C. Steiner Trees vs. MSTs

Traditionally net decomposition has been done using Minimal Spanning Tree (MST) algorithms, but fast and extremely accurate Rectilinear Steiner Minimal Tree (RSMT) construction algorithms have become increasingly popular in the literature [6], [32], [33]. FGR can use any well-formed net decomposition, so we study how the choice of net decomposition affects FGR's overall results—we compare MST to a combination of FLUTE [9] and FastSteiner [19] that returns

TABLE VI

COMPARING NET DECOMPOSITION BY MST VS. STEINER TREES ON THE ISPD ‘07 BENCHMARKS [18]. TIME TAKEN FOR DECOMPOSITION BY MST OR STEINER TREES IS LESS THAN 1 MINUTE ON ALL BENCHMARKS AND DOES NOT IMPACT RUNTIMES. WHILE USING STEINER TREE DECOMPOSITIONS RESULTS IN A REDUCTION IN ROUTED SEGMENT LENGTH OF 0.5%, IT INCREASES VIA COUNTS BY 1.8% AND THUS INCREASES THE TOTAL COST OF ROUTING SOLUTIONS BY 0.7%. DECOMPOSITION BY STEINER TREES INCREASES ROUTING TIME BY 22%.

Benchmark	Decomposition by MST				Decomposition by Steiner trees			
	Segment WL (e5)	Vias (e5)	Total cost	Time (m)	Segment WL (e5)	Vias (e5)	Total cost	Time (m)
adaptec1 2-d	35.88	6.19	54.44	451	35.78	6.24	54.49	403
adaptec1 3-d	36.37	17.36	88.45	430	36.26	18.04	90.37	395
adaptec2 2-d	33.21	6.36	52.30	56	33.10	6.43	52.38	170
adaptec2 3-d	33.74	18.72	89.89	64	33.62	19.37	91.72	168
adaptec3 2-d	96.09	11.60	130.89	179	95.55	11.67	130.57	222
adaptec3 3-d	97.02	34.21	199.66	243	96.42	35.49	202.90	281
adaptec4 2-d	90.02	11.66	125.00	19	89.37	11.72	124.53	18
adaptec4 3-d	91.28	30.56	182.96	55	90.59	31.59	185.35	58
adaptec5 2-d	102.79	16.45	152.13	713	102.56	16.63	152.45	771
adaptec5 3-d	103.89	52.03	259.98	740	103.62	53.78	264.97	796
newblue1 2-d	24.15	7.76	47.42	1441	24.00	7.74	47.22	1441
newblue1 3-d	24.15	23.37	94.26	1442	24.00	24.00	96.01	1442
newblue2 2-d	46.81	9.90	76.51	4	46.41	9.95	76.27	4
newblue2 3-d	47.91	28.08	132.16	10	47.51	29.08	134.75	10
newblue3 2-d	75.63	11.20	109.23	1555	75.24	11.15	108.71	1460
newblue3 3-d	75.63	32.69	173.71	1501	75.24	33.04	174.35	1462
Ratio					-0.52%	+1.81%	+0.74%	+22.0%

TABLE VII

COMPARING LAYER ASSIGNMENT WITH FULL 3-D ROUTING ON THE 3-D INSTANCES OF THE ISPD ‘07 BENCHMARKS [18]. TOTAL COST OF THE BETTER OF THE TWO SOLUTIONS (COMPARED FIRST BY OVERFLOW THEN BY TOTAL COST) FOR EACH BENCHMARK ARE HIGHLIGHTED IN BOLD.

Bench-mark		Layer Assignment					Full 3-d Routing				
		Total ovfl	Segment WL (e5)	Vias (e5)	Total cost	Time (m)	Total ovfl	Segment WL (e5)	Vias (e5)	Total cost	Time (m)
adaptec	#1	0	36.37	17.36	**88.45**	430	1456	36.02	17.55	88.70	1453
	#2	0	33.74	18.71	**89.89**	64	2	33.36	19.06	90.54	1444
	#3	0	97.02	34.21	**199.66**	243	2	96.69	34.77	201.01	1487
	#4	0	91.28	30.56	182.96	55	0	91.39	29.32	**179.36**	83
	#5	0	103.89	52.03	**259.98**	740	5512	102.78	52.27	259.61	1462
newblue	#1	514	24.15	23.37	**94.26**	1442	1012	24.21	22.33	91.19	1447
	#2	0	47.91	28.08	132.16	10	0	47.93	27.15	**129.40**	18
	#3	39828	75.63	32.69	**173.71**	1501	51098	75.73	29.30	163.63	1827

the better Steiner tree every time. FGR merges segments of decomposed nets, as described in Section III-D, and produces non-trivial Steiner trees even when given decompositions by MSTs. The results on the ISPD ‘07 benchmarks are shown in Table VI. Time taken for decomposition by MSTs or Steiner trees is less than 1 minute on all benchmarks and does not significantly impact runtimes. As expected, routed segment length is smaller when Steiner tree algorithms are used. On the other hand, using Steiner tree algorithms actually increases via counts by 1.8% and causes total cost to increase by 0.7%. All evidence we have seen suggests that MST decompositions leave more flexibility than minimum Steiner trees, allowing one to avoid some amount of detouring. Prior work has shown that optimal Steiner trees for a given set of points can vary widely, but specialized techniques can increase flexibility [4]. However, FLUTE and FastSteiner do not currently optimize tree flexibility. In addition, Steiner points may inadvertently be placed in congested areas by the Steiner tree constructor, causing increased congestion and detouring. Congestion-driven Steiner trees could be helpful in this context, but apparently MSTs already provide a good solution and can also be biased to avoid congestion.

D. Layer Assignment vs. Full 3-d Routing

Section III-F above describes that FGR performs 3-d routing by first flattening the routing instance onto a 2-d grid, routing the new 2-d problem instance, and then converting the 2-d solution into a 3-d solution by assigning layers to routed segments, adding vias as necessary. FGR is also capable of solving 3-d problems directly by using full 3-d maze routing, and in Table VII we compare both methods. It is readily apparent that full 3-d routing takes far longer than 2-d routing with layer assignment, most likely because 3-d routing is more complex. On the easiest benchmarks, adaptec4 and newblue2, full 3-d routing takes at least 50% longer, but is able to decrease via counts significantly and in turn improve total cost by 2.0% and 2.1%, respectively. On the other hand, on the benchmarks where FGR with layer assignment cannot find a legal solution within 24 hours, newblue1 and newblue3, full 3-d routing produces solutions with significantly more overflow given the same timeout.

V. CONCLUSIONS

In this paper we have presented FGR, a high-performance global router for nanometer scale designs. FGR’s implementation is very compact—core algorithms and data structures require only 1200 lines of C++ code, which is available for download from `http://vlsicad.eecs.umich.edu/BK/FGR/`. FGR outperforms the best results from the ISPD ‘07 Global Routing Contest, as well as previous literature, in terms of route completion, runtime and total wirelength. In particular, FGR improves upon wirelengths produced by BoxRouter 1.0 and MaizeRouter in March 2007 by 10.6% and 9.1%, respectively. Comparing FGR 1.1 to most recent routers, FGR 1.1 outperforms Archer in wirelength by 10.1% and BoxRouter 2.0 by 4.9%. FGR is likely to boost research in physical design, while also leading to better commercial place-and-route tools [14].

The superficial similarity between negotiated-congestion routing and Lagrangian relaxation was known to the authors since 2001 and may have been observed by others, but we were unable to find any discussion in the literature. More importantly, by formulating Lagrange multipliers and explicitly deriving cost updates, our work demonstrates a discrepancy between the two approaches. The Lagrangian approach to large-scale routing explains the last-gasp problem and shows why multi-layer routing formulations are more difficult to solve directly than two-dimensional routing. While negotiated-congestion routing does not specify how to handle net weights, we derive the necessary formulas for using net weights with Lagrange multipliers.

Another key challenge is to integrate accurate congestion modeling provided by FGR into global and detail placement. This could be used to mitigate congestion early and provide accurate information about length of individual wires, which is particularly important in timing-driven placement.

FGR's core algorithms are directly relevant to detail routing of ASICs and FPGAs, while its constraint-driven nature makes it amenable to the handling of complex design rules. To this end, a key challenge for future research is to develop a prototype of a detail routing tool based on negotiated-congestion routing. Such a prototype would be particularly useful to explore design rules and models expected at future technology nodes as well as manufacturability and yield optimization which are a focus of modern industrial tools.

Acknowledgments. This work was partially supported by the National Science Foundation (NSF) and the Horace H. Rackham School of Graduate Studies at the University of Michigan.

References

[1] S. N. Adya, I. L. Markov and P. G. Villarrubia, "On Whitespace and Stability in Physical Synthesis," *Integration: the VLSI Journal* 39(4), pp. 340-362, 2006.

[2] C. Albrecht, "Global Routing by New Approximations for Multicommodity Flow," *IEEE TCAD* 20(5), pp. 622-632, 2001.

[3] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *FPGA*, pp. 213-222, 1997.

[4] E. Bozorgzadeh, R. Kastner and M. Sarrafzadeh, "Creating and Exploiting Flexibility in Rectilinear Steiner Trees," *IEEE TCAD* 22(5), pp. 605-615, 2003.

[5] T. Chan, J. Cong, T. Kong and J. Shinnerl, "Multilevel Optimization for Large-scale Circuit Placement," In Proc. *ICCAD*, pp. 171-176, 2000.

[6] M. Cho and D. Z. Pan, "BoxRouter: A New Global Router Based on Box Expansion and Progressive ILP," In Proc. *DAC*, pp. 373-378, 2006.

[7] M. Cho, K. Lu, K. Yuan and D. Z. Pan, "BoxRouter 2.0: Architecture and Implementation of a Hybrid and Robust Global Router," *ICCAD*, pp. 503-508, 2007.

[8] M. Cho, H. Xiang, R. Puri and D. Z. Pan, "Wire Density Driven Global Routing for CMP Variation and Timing," In Proc. *ICCAD*, pp. 487-492, 2006.

[9] C. C. N. Chu and Y.-C. Wong, "FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design," *IEEE TCAD* 27(1), pp. 70-83, 2008.

[10] J. Cong, J. Fang and Y. Zhang, "Multilevel Approach to Full-Chip Gridless Routing," In Proc. *ICCAD*, pp. 396-403, 2001.

[11] J. Cong, M. Xie and Y. Zhang, "MARS—A Multilevel Full-Chip Gridless Routing System," *IEEE TCAD* 24(3), pp. 382-394, 2005.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. Section 24.3: Dijkstra's algorithm, pp.595-601.

[13] W. A. Dees, Jr. and P. G. Karger, "Automated Rip-up and Reroute Techniques," In Proc. *DAC*, pp. 432-439, 1982.

[14] R. Goering, "IC Routing Contest Boosts CAD Research," EE Times, March 22, 2007. `http://www.eetimes.com/showArticle.jhtml?articleID=198500084`

[15] R. Hadsell and P. H. Madden, "Improved Global Routing through Congestion Estimation," In Proc. *DAC*, pp. 28-34, 2003.

[16] J. Hu and S. S. Sapatnekar, "A Survey on Multi-net Global Routing for Integrated Circuits," *Integration, the VLSI Journal* 31(1), pp. 1-49, 2001.

[17] ISPD 1998 Global Routing benchmark suite. `http://www.ece.ucsb.edu/~kastner/labyrinth`

[18] ISPD 2007 Global Routing Contest and benchmark suite. `http://www.ispd.cc/ispd07_contest.html`

[19] A. B. Kahng, I. I. Mandoiu and A. Zelikovsky, "Highly Scalable Algorithms for Rectilinear and Octilinear Steiner Trees," In Proc. *ASP-DAC*, pp. 827-833, 2003.

[20] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Domain," *IEEE TVLSI* 7(1), pp. 69-79, 1999.

[21] R. Kastner, E. Bozorgzadeh and M. Sarrafzadeh, "Pattern Routing: Use and Theory for Increasing Predictability and Avoiding Coupling," *IEEE TCAD* 21(7), pp. 777-790, 2002.

[22] Lagrange multipliers. `http://en.wikipedia.org/wiki/Lagrange_multipliers`

[23] K.-Y. Lee and T.-C. Wang, "Post-routing Redundant Via Insertion for Yield/Reliability Improvement," In Proc. *ASP-DAC*, pp. 303-308, 2006.

[24] S. Lee and M. D. F. Wong, "Timing-driven Routing for FPGAs based on Lagrangian Relaxation," *IEEE TCAD*, pp. 506-510, 2003.

[25] C.-W. Lin et al., "Recent Research and Emerging Challenges in Physical Design for Manufacturability/Reliability," In Proc. *ASP-DAC*, pp. 238-243, 2007.

[26] C.-W. Lin et al., "Efficient Obstacle-avoiding Rectilinear Steiner Tree Construction," In Proc. *ISPD*, pp. 127-134, 2007.

[27] D. McGrath, "Routing Technology Came from Within Cadence, execs say," EE Times, Sept. 8, 2006. `http://www.eetimes.com/showArticle.jhtml?articleID=192700243`

[28] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-based Performance-driven Router for FPGAs," In Proc. *ACM Symp. on FPGAs*, pp. 111-117, 1995.

[29] M. Moffitt, Personal communication, March 2007.

[30] M. M. Ozdal and M. D. F. Wong, "A Length-matching Routing Algorithm for High-performance Printed Circuit Boards," *IEEE TCAD* 25(12), pp. 2784-2794, 2006.

[31] M. M. Ozdal and M. D. F. Wong, "Archer: A History-driven Global Routing Algorithm," *ICCAD*, pp. 488-495, 2007.

[32] M. Pan and C. Chu, "FastRoute: A Step to Integrate Global Routing into Placement," In Proc. *ICCAD*, pp. 464-471, 2006.

[33] M. Pan and C. Chu, "FastRoute 2.0: A High-quality and Efficient Global Router," In Proc. *ASP-DAC*, pp. 250-255, 2007.

[34] H. Ren, D. Z. Pan and P. G. Villarubia, "True Crosstalk Aware Incremental Placement with Noise Map," In Proc. *ICCAD*, pp. 402-409, 2004.

[35] J. A. Roy and I. L. Markov, "Seeing the Forest and the Trees: Steiner Wirelength Optimization in Placement," *IEEE TCAD* 26(4), pp. 632-644, 2007.

[36] J. A. Roy and I. L. Markov, "High-Performance Routing at the Nanometer Scale," *ICCAD*, pp. 496-502, 2007.

[37] L. K. Scheffer, "Physical CAD Changes to Incorporate Design for Lithography and Manufacturability," In Proc. *ASP-DAC*, pp. 768-773, 2004.

[38] L. K. Scheffer, L. Lavagno and G. Martin, eds., *Electronic Design Automation for Integrated Circuits Handbook*, CRC Press, 2006.

[39] Z. Shen, C. C. N. Chu and Y. M. Li, "Efficient Rectilinear Steiner Tree Construction with Rectilinear Blockages," In Proc. *ICCD*, pp. 38-44, 2005.

[40] K. So, "Solving Hard Instances of FPGA Routing with a Congestion-Optimal Restrained-Norm Path Search Space," In Proc. *ISPD*, pp. 151-158, 2007.

[41] P. V. Srinivas et al., "System and Method for Estimating Capacitance of Wires Based on Congestion Information," U.S. Patent 6519745, filed May 26, 2000, issued Feb. 11, 2003.

[42] D. C. Wang, "Method for Estimating Routability and Congestion in a Cell Placement for Integrated Circuit Chip," U.S. Patent 5587923, filed Sept. 7, 1994, issued Dec. 24, 1996.

[43] J. Westra, C. Bartels and P. Groeneveld, "Probabilistic Congestion Prediction," In Proc. *ISPD*, pp. 204-209, 2004.

[44] J. Westra and P. Groeneveld, "Is Probabilistic Congestion Estimation Worthwhile?", In Proc. *SLIP*, pp. 99-106, 2005.