Completing High-quality Global Routes

Jin Hu
University of Michigan
jinhu@eecs.umich.edu

Jarrod Roy
IBM Austin Research Lab.
jaroy@us.ibm.com

Igor Markov
University of Michigan
imarkov@eecs.umich.edu

ABSTRACT

To ensure chip manufacturability, all routes must be completed without violations. Furthermore, the chip's power consumption and performance are determined by the length of its routed wires. Therefore, our work focuses on minimizing wirelength. Our key innovations include: (1) a novel branch-free representation (BFR) for routed nets, (2) a trigonometric penalty function (TPF), (3) dynamic adjustment of Lagrange multipliers (DALM), (4) cyclic net locking (CNL), and (5) aggressive lower-bound estimates (ALBE) for A*-search, resulting in faster routing. We complete all routable ISPD 2008 contest benchmarks and re-placed *adaptec* suite without violation and produce shorter routes.

Categories and Subject Descriptors

J.6 [Computer-aided Engineering]: Computer-aided design (CAD)

General Terms

Algorithms, Design, Performance

Keywords

Global Routing

1. INTRODUCTION

As chip complexity grows, back-end tools must limit routed interconnect length, as this greatly affects the chip's performance, dynamic power, and yield. Moreover, violation-free global routing solutions facilitates smooth transitions to design-for-manufacture (DFM) optimizations. Conversely, solutions with violations imply the design will not function correctly when fabricated.

If a *global router* produces a violation-free (legal) solution, then the design can move onto detail routing and continue through the design process. However, if a routed design is inevitably unroutable or has violations, then a secondary step must isolate problematic regions (see Figure 1). Given a significant number of violations, it is common practice to fix the routing by repeating global and/or detail placement and injecting whitespace into congested regions. This type of congestion-driven placement is supported by both commercial and academic software, like Kraftwerk2 [20], and NTUPlace [9]; Capo with ROOSTER [19] and FastPlace with IPR [16] explicitly consider specific types of routes during global placement. *In other words, the global router is not solely responsible for producing a violation-free solution.*

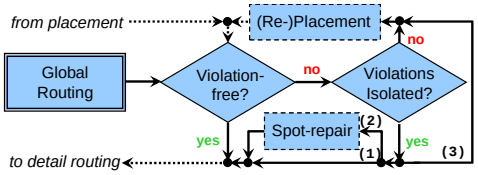

Figure 1: The global routing portion of the VLSI design flow. Fully routable designs are handed off to detail routing. Otherwise, the design can (1) be sent directly to detail routing, (2) go through spot-repair, or (3) go through re-placement iterations, depending on the severity of violations.

If the number of violations is small or the violations are sufficiently separable, then 1) a secondary tool can attempt to spot-repair the slightly illegal layout, 2) the design can be handed off to detail routing, or 3) the design is sent back to placement. Spot-repair is the most attractive option, as it allows the violations to be fixed without affecting the large majority of global routes. With a small number of violations, most commercial tools gamble on detail routing to resolve them. Therefore, a global router does not always *need* to minimize violations but it usually *must* minimize the total wirelength of the design because (i) the length of the routed nets directly affects how and if violations can be repaired, (ii) spot-repair does not significantly alter the total wirelength, and (iii) detail routing largely follows global routes.

In practice, even a small number of global-routing violations imply a *long runtime* in detail routing, *degraded signal integrity* caused by densely packed wires, and *dishing effects* caused by CMP during fabrication. Instead, designers allocate greater amounts of whitespace to wire-dense blocks during floorplanning while EDA tools use congestion-mitigation techniques during placement. Tools like FastRoute [15] were originally designed to provide congestion feedback to global placers [16] rather than as a high-quality router.

High-quality routing solutions on recent large-scale benchmarks [7] from IBM were produced by FGR 1.1 [18]. At the ISPD 2008 Global Routing Contest [8], NTHU-Route 2.0 [2] and NTUgr [3] also posted high-quality results, along with improved runtimes. In addition, FastRoute 4.0 [21] claimed exceptionally low runtimes.

Similar to [2, 3, 18], our work focuses on finding high-quality routing solutions. We note that decreasing high violation counts on unroutable benchmarks by less than 50% offers only marginal benefits because such situations typically require the design to be sent back to global placement [9, 16, 19, 20] or netlist restructuring.

ISPD'10, March 14–17, 2010, San Francisco, California, USA.

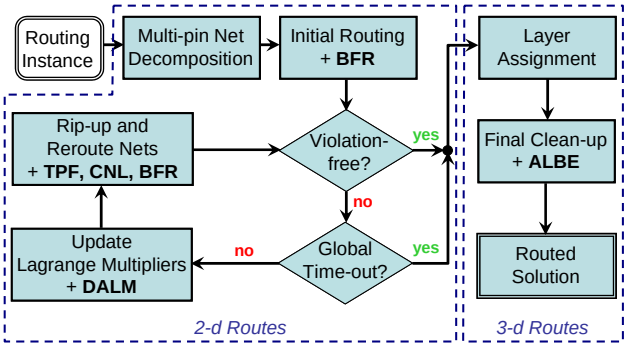

Figure 2: The flow of global routing in BFG-R and the use of novel techniques such as a branch-free representation (BFR) for routed nets, cyclic net locking (CNL), dynamic adjustment of Lagrange multipliers (DALM) a trigonometric penalty function (TPF), and aggressive lower-bound estimates (ALBE).

We make the following key contributions through BFG-R:

- Several improvements to existing routing algorithms that reduce wirelength, e.g., dynamic adjustment of Lagrange multipliers (DALM) and accurate 2-d via pricing.
- Reducing runtime by cyclic net locking (CNL).
- Techniques to reliably complete (without violations) designs such as an effective trigonometric penalty function (TPF).
- A branch-free representation (BFR) for single routed nets.
- An aggressive lower-bound estimate (ALBE) for A*-search.
- Empirical comparisons against the winners of the ISPD 2008 Global Routing Contest [8]. BFG-R completes the twelve routable ISPD 2008 Contest benchmarks without violations, more than any other router. On those benchmarks, BFG-R improves upon the solutions that were generated by NTHU-Route 2.0 [2], producing more violation-free solutions with comparable wirelength. BFG-R also produces better solutions than NTUgr [3] and FastRoute 4.0 [21] on the majority of designs. On a new set of benchmarks using re-placed *adaptec* netlists, we successfully route all designs without violation whereas all other routers fail on at least one design.

The remainder of this paper is structured as follows. Section 2 outlines BFG-R's global routing flow. Section 3 describes the new algorithms that are key to BFG-R's high performance. Section 4 discusses the data structure that allows BFG-R to maintain scalability for large problem instances and handle large netlists. Section 5 presents BFG-R's results on the ISPD08 Contest benchmarks [8] and *adaptec* netlists re-placed with mPl6 [1]. Section 6 concludes our current work and discusses future work.

2. GLOBAL ROUTING FRAMEWORK

In this section, we explain the general global routing framework of BFG-R, as shown in Figure 2. We also describe several key algorithms in BFG-R that significantly improve solution quality.

Given a global routing instance, BFG-R first splits multi-pin nets, nets with three or more pins, into two-pin subnets. BFG-R then produces an initial routing solution on a 2-d grid. If the design has no violations, BFG-R performs layer assignment – projecting 2-d routes onto a 3-d grid – and a final clean-up pass to minimize wirelength. If the design has violations and a global time-out has not been exceeded, then the Lagrange multipliers, factors that affect the edge cost, are updated. BFG-R then rips up any violating nets and reroutes them based on the costs of individual edges in a pre-determined order. This iterative process continues until either all violations have been resolved or the global time-out has expired. BFG-R then performs layer assignment and a final clean-up pass.

2.1 Multi-pin Net Decomposition

Competitive routers explicitly decompose (split) large nets into sets of two-pin subnets. There are two mainstream methods: (1) minimal spanning tree (MSTs), used by NTUgr [3] and FGR 1.1 [18], and (2) Steiner minimal trees (SMTs), used by NTHU-Route 2.0 [2] and FastRoute 4.0 [21]. Steiner trees offer minimal wirelength for nets and can therefore ease initial routing iterations. However, routers must support effective net restructuring, which requires advanced algorithms and flexible data structures. MST-based decompositions, on the other hand, can lead to a worse initial routing solution, as MSTs can have up to 150% of Steiner tree wirelength. Thus, the maze router must work harder to reduce wirelength and congestion. However, as we show in Section 4.1, subnets generated using MSTs can share resources and can be restructured into SMTs without explicitly storing branching points. Thus, BFG-R decomposes multi-pin nets using MSTs instead of SMTs. Second, it facilitates a stand-alone implementation and does not rely on external Steiner-tree packages.

2.2 Balancing Wirelength and Violations

A major challenge in large-scale routing is balancing wirelength against violations as competing objective functions. Published routers include separate factors to balance wirelength and congestion [14, Section 3.4] by tuning weights in linear combinations. However, as stated in [4], *ad hoc* trade-offs may lead to violent divergence of routing iterations. Therefore, several routers use *dampening* factors to ensure convergence [4].

Instead of explicitly trading off wirelength for violations, BFG-R uses Lagrange multipliers with a complementary cost function. This approach effectively guides the routes to areas with lower cost and smaller congestion. This key technique, *edge*-centric Lagrange multipliers, was introduced first in [18]. While Lagrangian relaxation has been suggested for global routing, previous work has either been specific to timing-driven routing and maintain *net*-centric Lagrange multipliers [10] or focused on a single net at a time. These algorithms use conventional history-based rip-up and reroute for the router's main loop. In contrast, our formulation directly handles modern instances of global routing, such as those from the ISPD '07 and '08 contests. Unlike previous routers, the history cost is only based on the congestion and does not affect the base cost of a routing edge. The cost of an edge e depends on its base cost b_e, Lagrange multiplier h_e, and congestion penalty p_e [18]:

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

Lagrange multipliers are updated at the beginning of rip-up and reroute iteration k in the following way [18]:

$$h_e^k = \begin{cases} h_e^{k-1} + h_{step} & \text{if } e \text{ is over-capacity} \\ h_e^{k-1} & \text{otherwise} \end{cases} \tag{2}$$

Compared to previous work, we use a different penalty function p_e for local congestion (see Section 3.3), and we do not use a constant h_{step} (see Section 3.2). The stopping criterion for rip-up and reroute iterations gauges the amount of effort applied on hard-to-route instances. In our current implementation, the default version of BFG-R stops when a legal solution is found or upon running for 24 hours (according to the ISPD08 routing contest).

2.3 Net Ordering

Nets that use over-capacity edges are ripped up and must be rerouted. We have observed the best results when subnets were routed (i) in ascending order of their bounding box area in areas of *low congestion* and (ii) in ascending order of how much their bounding box area deviates from the median bounding box area in areas of *high congestion*.

2.4 Point-to-point Routing

During *initial routing* and *rip-up and reroute* (R&R), a router must connect pin pairs in the routing grid. Common methods include pattern routing, used by NTHU-Route 2.0 [2] and Sidewinder [6], and monotonic maze routing, used by FastRoute 2.0 [16].

In pattern routing, a small number of route shapes are examined to connect the points. Typically, these shapes have short wirelength and few bends such as L, U, and Z patterns. This method is the fastest method to connect pin pairs, especially when there are no routing obstacles or over-capacity routing edges present. In practice, we notice that about 90% of subnets from the final routing solution are L-shaped (includes flat subnets). However, in the presence of congestion, the vast majority of runtime is spent routing connections that are not pattern-shaped.

If there are relatively few obstacles, monotonic maze routing is a viable option to route two-pin subnets. Instead of following a set path, the monotonic router searches those edges that move closer to the target in terms of Manhattan distance. Monotonic routing can be performed in linear time, using dynamic programming [17]. This method finds any route that pattern-routing can, but, due to decision making overhead, will take longer.

If there are more than a few blockages, monotonic routing can fail to find a violation-free route, even if one exists. Instead, a better alternative is to use boxed A*-search (BAS) with an accurate lower-bound function. BAS (1) combines Dijkstra's shortest path algorithm with a non-trivial lower-bound function[1], (2) restricts the search space to within the pins' bounding box (or a wider box), and (3) allows all edges to be traversed anytime during the search. BAS finds the solution with minimal detouring, given that a path exists. Routes that are found by pattern and monotonic routing are a proper subset of those found by BAS, but using BAS for those routes usually takes longer.

2.5 Continuous Net Restructuring

Published competitive routers, NTHU-Route 2.0 [2], NTUgr [3], FastRoute 4.0 [21] and FGR 1.1 [18], all employ net restructuring during maze routing. To preserve topological flexibility, we restructure nets continually similar to FGR 1.1. To limit runtimes, we developed a new technique called dyclic Net Locking (CNL), described in Section 3.5 below.

2.6 End-game Optimizations

After rip-up and reroute, BFG-R performs layer assignment followed by a final clean-up on the 3-d grid. There are two basic approaches to layer assignment. The simplest, but impractical, approach is to use maze routing on the entire 3-d routing grid. The more common approach, used by nearly all competitive routers, starts by compacting the 3-d routing grid onto a simpler 2-d grid with aggregated routing resources. The search is then performed on the 2-d grid. After maze routing finishes, 3-d routes for each net are reconstructed from solutions obtained from the 2-d grid.

The authors of [18] show that if edge capacities are aggregated properly, there exists a 3-d solution that has the same number of violations as the 2-d solution. Several methods to assign the routes to layers have been proposed, including an ILP-based algorithm [4], dynamic programming [11], and a greedy approach [18]. BFG-R's layer assignment adapts a fast, greedy strategy followed by one round of full 3-d wirelength reduction.

After layer assignment and before traditional 3-d clean-up, we iterate over all routing edges and temporarily increase the capacities of edges with violations so that they become 100% utilized. This makes the solution temporarily legal. Next, we apply the clean-up pass and find alternative shorter routes. Note that the total and maximum overflow of the original 2-d solution cannot increase, as (1) edges that have violations are already illegal and (2) edges that have no violations cannot become illegal.

After clean-up, we reinstate the correct capacities for all routing edges and recalculate the total and maximum overflow for the final solution. We observe that this clean-up method is as effective in reducing total wirelength usage in illegal solutions as it is in legal solutions, and usually decreases total overflow by a small amount.

[1]Vias are not represented explicitly, but priced implicitly.

3. KEY ALGORITHMS IN BFG-R

In this section, we outline key enhancements to our global routing flow that improve the router's overall performance and its ability to quickly find high-quality solutions.

3.1 Edge Clustering During Rip-up

To improve memory locality and cache utilization, BFG-R first finds all edges that have at least one violation and clusters them based on location. To this end, BFG-R starts with an arbitrary edge and performs a breadth-first expansion through neighboring edges. Over-capacity neighboring edges are added to the current cluster, and the expansion continues. After collecting all over-capacity edges in the area, BFG-R finds the next over-capacity edge and initiates a new cluster. This process continues until all over-capacity edges are clustered. Each edge will only belong to exactly one cluster, and performing R&R by clusters will not require extra work. Moreover, we ensure that if a subnet crosses multiple clusters, it will only be ripped up and rerouted once.

BFG-R then considers each cluster in order of increasing violation count. That is, it first rips up and reroutes the nets in relatively uncongested areas in hopes of freeing up valuable resources for the more congested clusters. After the first few R&R iterations, when congested edges break down into separate regions, edge clustering roughly halves the runtime of subsequent iterations.

3.2 Dynamically Adjusting Lagrange Multipliers (DALM)

Numerical updates of Lagrange multipliers (history costs) are critical to the success of the negotiated-congestion [12] and discrete Lagrange multiplier [18] routing frameworks. They must be precisely determined since they are the dominant factors in determining both solution quality and runtime.

Previous work [12, 18] increases Lagrange multipliers of congested edges by a constant h_{step} according to Equation 2. Empirically, we find that large steps lead to increased speed but also increased detouring. Conversely, small steps lead to lower final wirelength but much increased runtime. Further complicating the issue is that different benchmarks have drastically different optimal ranges of steps. Therefore, we use the following two, more aggressive, history cost functions to balance runtime and quality:

$$h_e^k = \begin{cases} h_e^{k-1} + h_{step} \times 1.25 & \text{if } e_{OF} \geq TEdge_{OF} \\ h_e^{k-1} + h_{step} & \text{else if } e_{OF} > 0 \\ h_e^{k-1} & \text{otherwise} \end{cases} \quad (3)$$

where $TEdge_{OF} = \max(e_{OF}) \times 95\%$. That is, all edges that have overflow within 5% of the maximum edge overflow will have an additional increase to its history cost. Otherwise, an overflown edge will receive the standard increment. For the largest cluster, we give the most congested edges an additional cost:

$$h_e^k = h_e^k + (1 - cluRatio + \alpha)^{-1} \text{ if } e_{OF} \geq TClu_{OF} \quad (4)$$

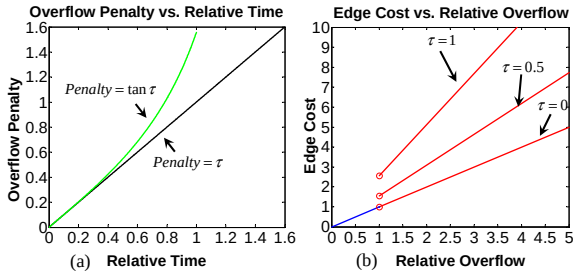

Figure 3: Trigonometric cost function used in BFG-R. The overflow penalty grows trigonometrically with the relative time τ (a). The cost function grows linearly with overflow (b).

where $cluRatio$ is the cluster size divided by the total number of over-capacity edges, $TEdge_{OF} = \max(clu_{OF}) \times 90\%$, and $0 \leq \alpha < 1$. That is, within the largest (and most congested) cluster of over-capacity edges, for the edges within the top 10% of the maximum edge overflow within the cluster will receive an additional increase. The parameter α controls how fast the penalty should grow. In practice, we have found that $\alpha = 0.75$ works well to balance solution quality and runtime.

To find better Lagrange steps, we adjust them dynamically between iterations of rip-up and reroute. We allow for a generous range of Lagrange steps, which includes the optimal range of all available benchmarks, and adapt the step within $[h^{min}_{step}, h^{max}_{step}]$ over time. Our initial step is chosen to be $\frac{h^{max}_{step}+h^{min}_{step}}{2}$, and we choose a *delta* for Lagrange steps $\Delta_{step} = \frac{h^{max}_{step}-h^{min}_{step}}{200}$. We route in the framework of Section 2 and Figure 2, while Lagrange steps are modified between iterations as follows

$$h^{k+1}_{step} = \begin{cases} h^k_{step} + \Delta_{step} & \text{if } viol_k \geq viol_{k-1} \\ h^k_{step} - \Delta_{step} & \text{if } viol_k < viol_{k-1} \text{ and } WL_k > WL_{k-1} \\ h^k_{step} & \text{if } viol_k < viol_{k-1} \text{ and } WL_k \leq WL_{k-1} \end{cases} \quad (5)$$

Empirically, Lagrange steps change significantly during the early iterations of rip-up and reroute, settle to within a small range of steps during the middle iterations, and finally increase when nearing a legal solution. As reported in Table 1, this technique helps BFG-R find high-quality routes while reducing violation counts.

3.3 Trigonometric Penalty Function (TPF)

A competitive router must ensure that its iterations make consistent progress. If the benchmark is routable, a global router should eventually find a solution with no overflow. However, routers can take a long time to clear the last few violations on difficult-to-route (but routable) designs; lack of progress can force a router into a local minimum. This situation is magnified in the benchmark *newblue1*, where several routers struggle to find a legal solution.

To find better routes, other routers increase the penalty for overflow over time. For instance, Hadsell et al. [5] amplified the congestion cost at a linear rate, capping the growth at 1.2× after the routing edge was over- capacity by 20%; FGR 1.1 [18] and NTHU 2.0 [2] increased the penalty for overflow at an exponential rate over time. However, an overly sharp or discontinuous increase in penalty may mislead the maze router early on and cause it to find poor-quality routes. Therefore, the penalty function must continuously increase, starting at low values.

We propose a new penalty function p of a routing edge e based on its relative overflow ω_e and the relative time $\tau = \frac{CurrentTime}{MaxTimeAllowed}$

$$p(e) = \begin{cases} \omega_e \times (1 + \tan(\tau)) & \text{if } \omega_e > 1 \\ \omega_e & \text{otherwise} \end{cases} \quad (6)$$

BFG-R's cost function grows linearly with overflow but trigonometrically with time, as shown in Figure 3. Note that toward the beginning, the growth factor is close to 0. Thus, it does not interfere with the original performance of the maze router. As runtime increases, the penalty grows faster in order to properly direct the maze router to find violation-free routes. In practice, BFG-R is able to legally route *newblue1* (without violations) while solutions found by other routers have violations and higher wirelength.

3.4 Via Pricing

To perform 3-d routing, BFG-R first generates the routes on the 2-d grid and then projects the routes onto the 3-d grid. Thus, during 2-d routing, a global router should be aware of the cost to cross layers. The most common approach to price vias is to use a constant cost function. Some other routers have via cost decrease over time [2] or use benchmark-specific fixed costs [21].

During 2-d routing, BFG-R estimates the ratio between the number of 3-d vias to 2-d vias. That is, the expected number of 3-d vias needed to represent one 2-d via is proportional to the number of layers. Thus, to accurately model the number of 3-d vias needed per route, we price 2-d vias as follows

$$p(VIA) = \lceil l/2 \rceil \times viaFactor \quad (7)$$

where l is the number of available routing layers and $viaFactor$ is the original price of a 3-d via specified by the designer.

3.5 Cyclic Net Locking (CNL)

We observed through profiling that the vast majority of runtime in the unmodified BFG-R flow is spent routing nets with large bounding boxes. Since all violating nets will eventually be ripped up, we control how often long nets are ripped up.

BFG-R classifies subnets by the area of their bounding box measured in whole routing grid cells, or *GCells*, so that long flat subnets do not have zero area.

$$Area(\text{BBox}_n) = (|\text{BBox}_n.x1 - \text{BBox}_n.x2| + 1) \times (|\text{BBox}_n.y1 - \text{BBox}_n.y2| + 1) \quad (8)$$

This effectively estimates the search space for boxed A*-search on a subnet. From this, we found that (1) almost all of the nets' routes are within 2× of their HPWL and (2) few nets route with a significant number of detours.

Therefore, we propose to lock larger subnets after the first few iterations of rip-up and reroute, but unlock them periodically after. How often a subnet is unlocked is determined based on the size of its bounding box relative to the average bounding box size:

$$AvgArea = \left(\frac{1}{N}\right) \sum_{n=1}^{N} Area(\text{BBox}_n) \quad (9)$$

A subnet n is allowed to be rerouted every $Period(n)$ iterations:

$$Period(n) = \min\left\{ \left\lceil \frac{Area(\text{BBox}_n)}{AvgArea} \right\rceil, 20 \right\} \quad (10)$$

Thus large subnets are unlocked less frequently than small subnets (but at least every 20 iterations) and subnets with average or smaller area are never locked. We chose not to unlock many nets at once, but instead use a *dispersive* strategy that aims to unlock similar numbers of nets at each iteration. To do so, subnet n is allowed to be unlocked during iteration i if the following condition is satisfied

$$(i < 2) \text{ or } ((i + n) \mod Period(n) = 0) \quad (11)$$

This condition effectively staggers unlocking of large nets and also allows them to be unlocked with the proper period. We find that

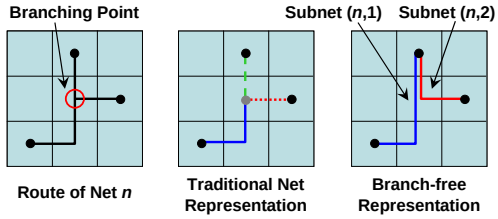

Figure 4: The branch-free representation (BFR) of routed nets. Subnets are treated separately but can share routing edges. Collectively they represent a Steiner tree.

this method improves the framework of Section 2 dramatically with little impact on solution quality[2]. The success of CNL indicates significant flexibility in choosing which nets to reroute, and justifies the focus on rerouting shorter nets for efficiency.

3.6 Aggressive Lower-bound Estimate (ALBE)

Of commonly-used point-to-point routing techniques, A*-search is the most flexible and guarantees to find the shortest path if a path exists. However, A*-search degenerates into Dijkstra's algorithm if its admissible function underestimates the true path-suffix cost too much. This effect is especially pronounced when 1) temporarily setting shared edge costs to zero when routing multi-pin subnets and 2) using traditional distance-based lower-bound functions, e.g., distance $\times$ cost of the cheapest edge, after history has accumulated. The growth of history costs hampers the maintenance of minimum edge costs in a given region, and routers typically do not increase the initial minimum edge cost as history costs accumulate.

In the presence of even a single zero-cost edge, the *minimum edge cost* becomes zero, and traditional distance-based admissible functions used for A*-search become trivial. To combat this, FGR 1.1 [18] and BFG-R employ ϵ-sharing, where shared edges are given a small $\epsilon > 0$ cost, rather than zero.

To maintain the speed of A*-search as history costs grow, we use an aggressive lower-bound estimate. For each subnet, instead of using the minimum edge cost of all possible edges to compute a distance-based lower bound, we traverse its path from the last iteration of R&R and use the minimum cost along that route. As per ϵ-sharing, each shared routing edge contributes less than a non-shared edge. Not only is this a more realistic method to estimate a lower bound of the new path, the search is sped up as it uses a greater lower-bound function.

One caveat with using this estimate is that it can be too high to serve as an admissible function. That is, this estimate can slightly over-estimate the actual cost. When this happens, BAS maintains its speed but can (sometimes) overlook optimal routes. We therefore do not rely on aggressive lower bounds during R&R but use them to reduce the runtime of our greedy clean-up. In this context, its impact on solution quality is negligible.

4. ROUTE REPRESENTATION

High-performance routing demands transparent data structures. What and how to store is equally important compared to what *not* to store, as excessive sophistication of data structures often leads to poor performance in practice. Compared to the top routers from the ISPD 2008 Contest, we use about the same amount of memory as FastRoute 4.0, 20% less than NTUgr, and 2.5$\times$ less than NTHU (4$\times$ less on the largest benchmarks).

[2]It is not difficult to *ensure* that approximately equal numbers of nets are routed per iteration using randomization, but our method is straightforward and works well in practice.

4.1 Branch-free Representation (BFR) of Individual Routed Nets

Several possible data structures can represent nets with three or more pins. The most straightforward approach is to divide each net into a group of disjoint line segments (with bends). In the case of the three-pin net n shown in Figure 4, this would add a branching or Steiner point to the middle of the net, creating three *segments*, a set of connected routing edges in one direction. This representation supports proper calculation of routing resources and is used in global routers such as FR 4.0 [16] and NTHU-R 2.0 [2]. Other routers like MaizeRouter [13] store only the full horizontal and vertical segments but no intermediate points. However, this representation severely limits net restructuring, which modern global routers frequently perform – either explicitly by decomposing nets or implicitly through maze routing as in FR 4.0 and FGR 1.1 [18]. The process of restructuring nets causes branching points to move, appear, and disappear, which is difficult to support. Once a net is restructured, segments or branching points must be internally modified, e.g., branching points added, larger segments split into smaller segments, to support the new topology.

We propose a different data structure where branching points are represented implicitly. Let a *subnet* be a pair of terminal pins of a *net*. For each *subnet*, we store (i) each *occupied routing edge*, and not segments, and (ii) the coordinates of its endpoints. These pairs of points must collectively form a spanning tree, e.g., a minimum spanning tree (MST). Each net also stores the indices of the routing edges it uses and can easily find its subnets that use a particular routing edge. Such a mapping can be implemented with an STL hash-map or balanced binary tree, but in practice both data structures require too much memory. Instead, our memory-efficient data structure is an array of pairs of (1) routing edge indices and (2) the number of subnets of the net that pass through the edge. This container supports $O(\log |E|)$-time search, and $O(|E|)$-time insertion and deletion, where $|E|$ is the number of edges. However, in practice, the number of traversed edges is small.

Since each net stores the indices of used edges, routing resource usage can be calculated exactly and efficiently. These data structures allow BFG-R to maintain Steiner trees for nets without an explicit representation of branching points. We also find that BFR can ease the implementation of a router, as branching points are processed implicitly during maze routing rather than being created and destroyed explicitly. We found that 1) the overlap in BFR between subnets is small, as long as the net is initially decomposed using an MST and 2) coalescing subnets takes little time. Other routers, on the other hand, choose to use more memory to reduce runtime. For example, NTHU-R [2] uses large hash maps and precomputes edge costs for constant-time look-up.

4.2 Supporting Efficient Rip-up and Reroute

To facilitate efficient rip-up and reroute (R&R), fast identification of which subnets should be ripped-up at each iteration is crucial. Furthermore, the process of finding the appropriate subnets must take negligible time. To this end, BFG-R stores a list of passing subnets every routing edge. To quickly determine which connections need to be adjusted during an iteration, BFG-R goes over all routing edges, finds which edges are over-capacity, and adds the subnets that use the edge to the list of subnets to be ripped up.

When ripping up subnet s, every routing edge e used by s removes s from its list of subnets. The map maintained by net is then updated to reflect that one of its subnets no longer uses e. If no other subnets of the same net use e, it is removed from the map and the resources are returned to the edge. Lastly, every routing edge e is removed from the list of used edges maintained by s.

Benchmark	NTHU-Route 2.0 [2]			NTUgr [3]			FastRoute 4.0 [21]			Best Tuned [2, 3, 21]			BFG-R (No Tuning)		
	OF total	Cost (e6)	Time (m)	OF total	Cost (e6)	Time (m)	OF total	Cost (e6)	Time (m)	OF total	Cost (e6)	Router Name	OF total	Cost (e6)	Time (m)
Solution Quality and Runtime for ROUTABLE Benchmarks															
adaptec1	0	5.37	6.4	0	5.67	42.4	0	5.50	3.6	0	5.36	NTHU 2.0	0	5.43	8.4
adaptec2	0	5.24	2.8	0	5.47	7.4	0	5.28	1.2	0	5.23	NTHU 2.0	0	5.23	3.7
adaptec3	0	13.15	4.2	0	13.77	35.0	0	13.26	2.7	0	13.11	NTHU 2.0	0	13.14	16.0
adaptec4	0	12.18	15.1	0	12.41	14.7	0	12.15	1.1	0	12.17	NTHU 2.0	0	12.16	5.2
adaptec5	0	15.54	5.2	0	16.52	100.9	0	15.91	10.3	0	15.54	NTHU 2.0	0	15.67	15.5
bigblue1	0	5.57	10.0	0	5.95	118.3	0	5.89	8.0	0	5.57	NTHU 2.0	0	5.72	10.2
bigblue2	86	9.00	12.2	118	9.47	212.0	Invalid Solution			0	9.06	NTHU 2.0	0	9.11	40.8
bigblue3	32	13.07	9.7	0	13.49	25.6	MAZE RIPUP WRONG			0	13.08	NTHU 2.0	0	13.18	20.6
newblue1	164	4.60	14.2	212	4.82	136.0	542	4.73	13.6	0	4.65	NTHU 2.0	0	4.68	256.9
newblue2	0	7.59	1.1	0	7.85	5.1	0	7.53	0.7	0	7.53	FR 4.0	0	7.57	1.5
newblue5	18	23.14	29.0	0	24.25	117.9	0	23.51	13.8	0	23.17	NTHU 2.0	0	23.30	47.6
newblue6	0	17.70	49.4	0	18.74	76.6	MAZE RIPUP WRONG			0	17.70	NTHU 2.0	0	18.01	15.7
Routing Failures	4			2			4			0			0		
Improv. 0 OF		0.99			1.04			1.01			0.99			1.00	
Solution Quality and Runtime for UNROUTABLE Benchmarks															
bigblue4	256	22.80	72.9	410	24.35	302.9	Invalid Solution			162	23.10	NTHU 2.0	434	23.20	1416.6
newblue3	Time Out			33636	11.00	163.6	38020	10.88	1344.1	31106	17.15	NTUgr	33900	10.64	1420.9
newblue4	222	12.89	31.2	284	13.89	223.3	212	13.16	27.7	138	13.04	NTHU 2.0	218	13.08	1413.3
newblue7	68	35.52	1284.6	906	36.91	1403.9	Invalid Solution			54	35.58	FR 4.0	606	35.21	1421.1

Table 1: BFG-R compared with leading routers on the ISPD08 benchmarks [8], where `NTHU 2.0` is NTHU-Route 2.0 and `FR 4.0` is FastRoute 4.0. Experimental setup is described in Section 5.1. `Invalid Solution` indicates disconnected nets. `MAZE RIPUP WRONG` is an internal error produced by FastRoute 4.0. `Time Out` indicates that the router did not produce a solution within 24 hours. Runtimes are not averaged because (1) some routers did not produce valid solutions on all benchmarks, (2) some routers did not succeed on routable benchmarks, and (3) benchmark solution quality varies significantly.

When adding a new route to a subnet s, a similar sequence of steps is performed *in reverse*. That is, for every edge e the new route uses, it is first added to the list of used edges maintained by s. Next, if no other subnets (of the same net) use e, the map maintained by the net is updated to reflect one of its subnets now uses e. Finally, every routing edge e adds s to its list of subnets.

5. EMPIRICAL EVALUATION

First, we describe our experimental setup and the sets of benchmarks used. Next, we compare our solution quality on those benchmarks against the top three performers from the ISPD 2008 Global Routing Contest [8].

5.1 Experimental Setup

Our single-threaded implementation of BFG-R is written in C++, self-contained and does not require any external libraries, source code, or data files. We compiled our code with g++ 4.3.2 to produce a 64-bit binary. All BFG-R runs were performed on a quad-core 2.83 GHz processor with 8 GB of RAM. To draw objective conclusions, we also ran all other routers on the same machine with the exception of two benchmarks, *newblue3* and *newblue7*, for NTHU-Route [2] due to exceptional memory requirements. Instead, we ran those two designs on a 2.93 GHz processor with 20 GB of memory. Source codes of NTHU-Route 2.0, NTUgr, and FastRoute 4.0 were made available by the respective authors under the CEDA-sponsored open-source release. We compiled NTHU-Route's C++ code using g++ 4.1.2, as it is currently incompatible with g++ 4.3.2. We used g++ 4.3.2 to compile NTUgr's [3] C++ code and gcc 4.3.2 to compile FastRoute 4.0's [21] C code.

To ensure the proper execution of existing routers, we reproduced all published solutions and runtimes for NTHU, NTUgr, and FastRoute. We found that all three routers tuned to benchmarks. For example, FastRoute 4.0 used a set of specific parameters based on the benchmark name, as shown below.

```
if((strstr(benchFile,
          "adaptec1.capo70.3d.35.50.90.gr")!= NULL))
  SLOPE=5; THRESH_M=30; ENLARGE=15;
  ESTEP1=10; ESTEP2=5; ESTEP3=5;
  CSTEP1=5; CSTEP2=5; CSTEP3=10;
  COSHEIGHT=4; VIA=4; A=1; L_afterSTOP=1;
  mazeSet = 2; goingLV = TRUE; updateType = 0; }
```

Similarly, NTHU-Route is invoked by a Perl script that uses a different set of parameters for each ISPD 2008 benchmark; NTUgr used the number of non-trivial nets to differentiate between benchmarks and ran tailored flows with pre-set thresholds.

For an objective comparison, we ran each router, including BFG-R, in its default mode, where the router used the same configuration for all benchmarks. To negate tuning to specific contest benchmarks, we made superficial changes to the benchmark files, such as rename *adaptec1* $\rightarrow$ *xXaxXx1*.

5.2 Benchmarks

We used two sets of benchmarks for comparison. The first set is the well-known ISPD 2008 Global Routing Contest benchmarks. For the second set, we reused the netlists from the *adaptec* suite and placed them using mPl6 [1], a global placer that achieved the best overall wirelength while observing density constraints in the ISPD 2006 Placement Contest. We tested every target density in increments of 10%, starting at 100%. The target densities selected (and reported in Table 2) are transitional values for which the benchmarks became routable – increasing the target density by 10% would lead to routability problems.

5.3 Comparison of Results

In our experiments, each router was configured with identical parameters for all benchmarks. Table 1 compares BFG-R's performance on ISPD 2008 Contest benchmarks. Similarly, Table 2 compares BFG-R's performance on the re-placed *adaptec* suite. Row `Improv.0 OF` showing other routers' performance normalized to BFG-R's when both routers produced a fully legal solution. We compare our solution quality to those of NTHU-Route 2.0 [2], NTUgr [3], and Fast-Route 4.0 [21]. From the first set of benchmarks, only 12 of the 16 total designs are demonstrably routable. That is, no router has produced a legal solution for the designs *bigblue4*, *newblue3*[3] *newblue4*, and *newblue7*. Every design in the second set was shown to be routable by at least one router.

[3] *newblue3*, is trivially unroutable, as it contains a pin connected to over 2200 nets, which is greater than the total wire capacity of the GCell containing that pin.

Benchmark	NTHU-Route 2.0 [2]			NTUgr [3]			FastRoute 4.0 [21]			BFG-R		
	OF total	Cost (e6)	Time (m)	OF total	Cost (e6)	Time (m)	OF total	Cost (e6)	Time (m)	OF total	Cost (e6)	Time (m)
adaptec1, 70%	0	4.62	7.2	0	4.83	73.2	184	5.01	26.4	0	4.68	9.8
adaptec2, 60%	0	5.29	0.9	0	5.48	3.7	0	5.31	0.6	0	5.28	2.2
adaptec3, 80%	38	12.16	19.4	28	12.88	470.0	616	12.74	183.1	0	12.15	27.2
adaptec4, 80%	0	10.50	2.3	0	10.75	9.1	10	10.61	4.8	0	10.49	3.2
adaptec5, 70%	4	13.91	25.2	0	14.44	347.8	628	14.49	50.6	0	13.98	32.6
Routing Failures	2			1			4			0		
Improv. 0 OF		1.00			1.03			1.01			1.00	

Table 2: BFG-R compared with leading routers on the re-placed *adaptec* benchmark suite. Each benchmark's netlist was placed using mPl6 with its corresponding target density. These benchmarks were not used during the development of the routers we evaluate.

Routability. On the contest benchmarks, BFG-R finds legal solutions for all twelve routable benchmarks, whereas NTHU-Route 2.0, NTUgr, and FastRoute 4.0 produce four, two, and four illegal or invalid solutions, respectively. In particular, for the design *newblue1*, BFG-R is able to find a low-cost, violation-free solution whereas NTHU, NTUgr, and FastRoute all produce solutions with violations. Solution costs produced by NTUgr and FastRoute 4.0 are also higher than those of BFG-R's violation-free solutions.

On the five re-placed benchmarks, BFG-R is able to route all designs without violation, whereas NTHU, NTUgr, and FR 4.0 had two, one, and four violating designs, respectively. In particular, BFG-R finds a legal solution on *adaptec3* with 80% target density with competitive wirelength when no other router could not.

Wirelength. As illustrated in Table 1, on average, BFG-R produces routes that are comparable to those of NTHU-R 2.0 and 4% better than NTUgr on the designs where routers produced violation-free solutions. BFG-R is 1% better than FastRoute 4.0, but the sample space is reduced by four designs, as FR 4.0 produced an invalid solution (having disconnected nets) for *bigblue2*, came up with an internal error `MAZE RIPUP WRONG` for *bigblue3* and *newblue6*, and generated a solution with violations for *newblue1*.

On the five re-placed benchmarks, BFG-R produces routes that are comparable to the three valid solutions from NTHU and the one valid solution produced by FastRoute. Out of the four valid solutions found by NTUgr, BFG-R runs much faster and finds solutions that are 2% better. In the majority of cases, BFG-R's violation-free solutions cost less than other routers' solutions with violations.

6. CONCLUSIONS AND FUTURE WORK

We have presented BFG-R, a robust and scalable global router that produces highest-quality solutions in comparison to NTHU-Route 2.0 [2], NTUgr [3], and FastRoute 4.0 [21]. We introduced a set of key techniques that significantly improve BFG-R's performance on routable benchmarks: a trigonometric penalty function (TPF), dynamic adjustment of Lagrange multipliers (DALM), cyclic net locking (CNL), and aggressive lower-bound estimates (ALBE). We introduced a branch-free representation (BFR) for routed nets to improve net flexibility. If a legal solution exists, the techniques proposed in this work ensure that a legal solution with competitive wirelength will be found.

We have shown that BFG-R can consistently produce a low-cost, legal routing solution, as long as the design is routable. In contrast, NTHU can route designs somewhat faster, but does not guarantee a legal solution unless given a pre-determined set of parameters. NTUgr produces the most legal solutions, but at the cost of wirelength and runtime. FastRoute 4.0 is several times faster on the contest benchmarks but produces relatively poor solutions. On the second set of benchmarks, FastRoute is unreliable, as it cannot find solutions to four out of five designs and often uses more runtime.

We have also explored the trade-offs made during implementation by different routers. *First*, a key difference between our implementation and other routers is the dynamic edge-cost computation and update. This feature is critical to support multiple routing pitches and wirewidths. *Second*, we have noticed that finding high-quality routes requires carefully adjusting Lagrange multipliers, which necessitates more iterations. *Third*, finding legal solutions requires a slowly increasing penalty for violations. *Fourth*, we have tried to incorporate pattern routing in our flow, but it has not improved our results.

Our current implementation does not explicitly target unroutable benchmarks, unlike competing routers. This is a major avenue for further improvement that we plan to pursue. We are also considering monotonic routing as a means to accelerate R&R iterations.

7. REFERENCES

[1] T. F. Chan, J. Cong, J. Shinnerl, K. Sze, M. Xie, "mPL6: Enhanced Multilevel Mixed-size Placement with Congestion Control," *Modern Circuit Placement*, pp. 247-288, 2007.

[2] Y.-J. Chang, Y.-T. Lee, T.-C. Wang, "NTHU-Route 2.0: A Fast and Stable Global Router," *ICCAD*, pp. 338-343, 2008.

[3] H.-Y. Chen, C.-H. Hsu, Y.-W. Chang, "High-performance Global Routing with Fast Overflow Reduction," *ASPDAC*, pp. 582-587, 2009.

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

[5] R. Hadsell, P. Madden, "Improved Global Routing through Congestion Estimation," *DAC*, pp. 28-31, 2003.

[6] J. Hu, J. A. Roy, I. L. Markov, "Sidewinder: A Scalable ILP-based Router," *SLIP*, pp. 73-80, 2008.

[7] ISPD 2007 Global Routing Contest and benchmark suite. `http://www.sigda.org/ispd2007/rcontest/`

[8] ISPD 2008 Global Routing Contest and benchmark suite. `http://www.sigda.org/ispd2008/contests/ispd08rc.html`

[9] Z.-W. Jiang, B.-Y. Su, Y.-W. Chang, "Routability-driven Analytic Placement by Net Overlapping Removal for Large-scale Mixed-size Designs," *DAC*, pp. 167-172, 2008.

[10] S. Lee, M. D. F. Wong, "Timing-driven Routing for FPGAs based on Lagrangian Relaxation," *IEEE TCAD*, vol. 22(4), pp. 506-510, 2003.

[11] T.-H. Lee, T.-C. Wang, "Robust Layer Assignment for Via Optimization in Multi-layer Global Routing," *ISPD*, pp. 159-166, 2009.

[12] L. McMurchie, C. Ebeling, "PathFinder: A Negotiation-based Performance-driven Router for FPGAs," *FPGA*, 1995.

[13] M. Moffitt, "MAIZEROUTER: Engineering an Effective Global Router," *IEEE TCAD*, vol. 27(11), pp. 2017-2026, 2008.

[14] D. Müller, "Optimizing Yield in Global Routing," *ICCAD*, pp. 480-486, 2006.

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

[16] M. Pan, C. Chu, "IPR: An Integrated Placement and Routing Algorithm," *DAC*, pp. 59-62, 2007.

[17] M. Pan, C. Chu, "FastRoute 2.0: A High-quality and Efficient Global Routing," *ASPDAC*, pp. 250-255, 2007.

[18] J. A. Roy, I. L. Markov, "High-performance Routing at the Nanometer Scale," *TCAD*, vol. 27(6), pp. 1066-1077, 2008.

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

[20] P. Spindler, U. Schlichtmann, F. M. Johannes, "Kraftwerk2 – A Fast Force-Directed Quadratic Placement Approach Using an Accurate Net Model," *TCAD*, vol. 27(8), pp. 1398-1411, 2008.

[21] Y. Xu, Y. Zhang, C. Chu, "FastRoute 4.0: Global Router with Efficient Via Minimization," *ASPDAC*, pp. 576-581, 2009.