High-performance Global Routing for Trillion-gate Systems-on-Chips

by

Jin Hu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2013

To my family and friends

ACKNOWLEDGMENTS

I am grateful to my advisor, Professor Igor Markov, for all the advice and countless ideas he gave me throughout my graduate career. He provided me with many opportunities to improve my research and teaching skills, and taught me the true meaning of academic dedication and perseverance.

I would like to thank all my colleagues for their helpful contributions. In particular, I would like to thank Jarrod Roy, who mentored me and gave me valuable advice during my first few years, and Myung-Chul Kim, who was my primary collaborator during my last few years. I would also like to thank all past and current students that I met in Professor Markov's group, including Hector Garcia, Dong-Jin Lee, Johann Knechtel, George Viamontes, Dave Papa, Smita Krishnaswamy, Steve Plaza and Kai-Hui Chang. I am also thankful to Professor Eli Bozorgzadeh, Love Singhal and Debjit Sinha. Without their encouragement, I most likely would not have pursued a doctorate degree.

I would like to thank my parents for their support. I would also like to thank all my friends that helped keep me sane throughout the years, and gave me the much-needed breaks and fun. Thanks to all my bridge partners, including Jonathan Fleischmann, Max Glick and Zach Scherr. Thanks to Jeff Hao, Eric Wucherer, Nate Derbinsky, Pradeep Muthukrishnan, Timur Alperovich, Ganesh Dasika, Perry Iverson, Drew DeOrio, Joe Greathouse, Andrea Pellegrini, Debapriya Chatterjee and Jason Clemons for great times and experiences.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table

ABSTRACT

High-performance Global Routing for Trillion-gate Systems-on-Chips

by
Jin Hu

Chair: Igor L. Markov

Due to aggressive transistor scaling, modern-day CMOS circuits have continually increased in both complexity and productivity. Modern semiconductor designs have narrower and more resistive wires, thereby shifting the performance bottleneck to interconnect delay. These trends considerably impact timing closure and call for improvements in high-performance physical design tools to keep pace with the current state of IC innovation. As leading-edge designs may incorporate tens of millions of gates, algorithm and software scalability are crucial to achieving reasonable turnaround time. Moreover, with decreasing device sizes, optimizing traditional objectives is no longer sufficient.

Our research focuses on (i) expanding the capabilities of standalone global routing, (ii) extending global routing for use in different design applications, and (iii) integrating routing within broader physical design optimizations and flows, e.g., congestion-driven placement. Our first global router relies on integer-linear programming (ILP), and can solve fairly large problem instances to optimality. Our second iterative global router relies on Lagrangian relaxation, where we relax the routing violation constraints to allowing routing overflow at a penalty. In both approaches, our desire is to give the router the maximum degree of freedom within a specified context. Empirically, both routers produce competitive results within a reasonable amount of runtime. To improve routability, we explore the incorporation of routing with placement, where the router estimates congestion and feeds this information to the placer. In turn, the emphasis on runtime is heightened, as the router will be invoked multiple times. Empirically, our placement-and-route framework significantly improves the final solution's routability than performing the steps sequentially. To further enhance routability-driven placement, we (i) leverage incrementality to generate fast and accurate congestion maps, and (ii) develop several techniques to relieve *cell-based* and *layout-based* congestion. To broaden the scope of routing, we integrate a global router in a chip-design flow that addresses the *buffer explosion* problem.

PART I

Introduction and Background

CHAPTER I

Routing in Trillion-gate ASICs

As the complexity of digital designs grows, automated ASIC design flows must also evolve to keep up such that the produced integrated circuits (ICs) can be optimized for metrics such as performance and power. Traditionally, device or gate delay dominated chip performance. However, at current technology nodes, the performance bottleneck has shifted to interconnect delay, as (i) device delays improve faster than interconnect delay, and (ii) the amount of interconnect grows superlinearly with respect to the number of components. A trillion-gate system would typically be partitioned into tens or hundreds of smaller blocks, where then each individual block would undergo physical design and physical synthesis optimizations. Some of these blocks include on-chip memories, analog and mixed-signal blocks, high-speed I/O, general processing cores, digital signal processors, Fast Fourier Transform cores, and other circuits that are beyond the scope of the dissertation. The remaining blocks contain up to tens of millions of logic gates, which are bundled into a smaller set of *standard cells* (e.g., on the order of five million). The locations of individual logic gates and CMOS transistors are computed by offsetting the locations of respective standard cells by fixed offsets from the standard-cell library. During physical design optimization, designers must determine the locations of these standard cells, as well as their connectivity. This often requires several iterations of *placement* and *routing*. While every step affects timing closure, *global routing* is one of the fundamental stages. Known to be NP-complete [62], global routing impacts circuit performance, power, and turnaround time. Routing determines the length and delay of critical paths and therefore directly affects design timing. The recent ISPD 2007 and ISPD 2008 Global Routing Contests [51, 84] facilitated the development of novel routing techniques and algorithms, and inspired the creation of many scalable academic routers. This routing progress in part enabled the viable use of global routing in other design-flow steps, such as evaluating intermediate global placement solutions [108].

1.1 Challenges in Global Routing

Given that any given region on the chip can support a limited number of routes, it is imperative that: (i) the full assignment of routes has no *violations*, i.e., no location is over-subscribed, (ii) the routes are assigned such that every route has sufficient but uses minimal routing resources, and (iii) the assignment process has reasonable runtime.

Removing routing violations. State-of-the-art physical design tools must limit routed interconnect lengths, as this greatly affects the chip's performance, dynamic power, and yield. Moreover, violation-free routing solutions facilitate smooth transition to design-for-manufacture (DFM) optimizations.

If a global router produces a violation-free (legal) solution, then the design is passed to *detailed routing* and continues through the design process. However, if a routed design is inevitably unroutable or has violations, then a secondary step must isolate problematic regions (Figure 1.1). Given a significant number of violations, it is common practice to fix the routing by repeating global and/or detailed placement and injecting whitespace into congested regions. This type of congestion-driven placement is supported by both commercial and academic software [24, 58, 94, 103]. *In other words, the global router is*

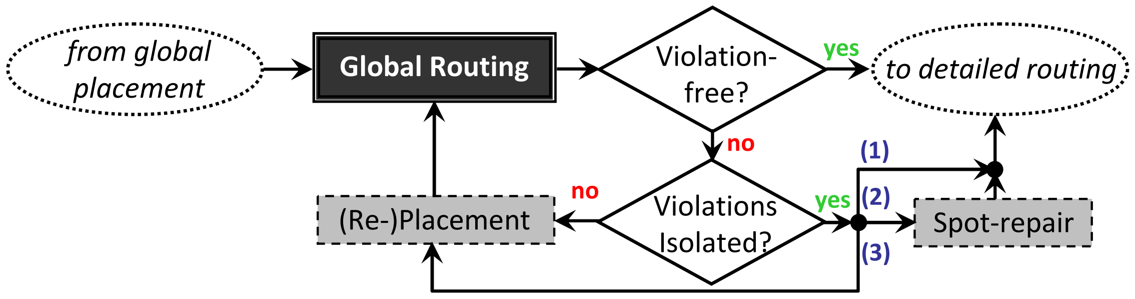


Figure 1.1: The global routing portion of the VLSI design flow. Fully routable designs are handed off to detailed routing. Otherwise, the design can (1) be sent directly to detailed routing, (2) go through spot-repair, or (3) go through re-placement iterations, depending on the severity of violations.

not solely responsible for producing a violation-free solution.

If the number of violations is small or the violations are isolated, then (1) a secondary tool can attempt to spot-repair the slightly illegal layout, (2) the design can be handed off to detailed 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 detailed 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) detailed routing largely follows global routes. In practice, even a small number of global-routing violations imply a *long runtime* in detailed routing, *degraded signal integrity* caused by densely packed wires, and *dishing effects* caused by *chemical mechanical polishing (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 [87] were intended to provide congestion feedback to global placers [24] rather than as a high-quality router.

Minimizing routed wirelength. Traditionally, in addition to producing a (near) violation-free routing solution, a global router's must also minimize *wirelength*, which is a combination of (i) the total number of *routing tracks* or *routing segments* used in each metal layer, and (ii) the total number of *vias*, i.e., connections in which to connect routing tracks across layers. However, with current technology scaling trends, designs are susceptible to coupling capacitance and other parasitic effects. Traversing from one metal layer to another is becoming costly as vias have non-trivial effects because they impact timing and may block several routing tracks [106]. In this respect, routing is even more important, as it directly determines the locations of the routes, as well as the number of vias. Thus, a router must also limit the number of vias as well as minimize the number of routing tracks.

Integrating routing and placement. In earlier technology generations, placement and routing algorithms were designed and implemented in separate software tools, even when the user interface exposed a single optimization to chip designers. Yet, common placement metrics no longer capture key aspects of solution quality at new technology nodes [4, 94]. Wirelength-optimized placements often lead to routing failures when the placer is not aware of actual routes [24]. Prior work incorporates *routing congestion* analysis, i.e., the ratio between route usage and route capacity, into global placement, but lacks in several aspects. First, simplified congestion models do not capture phenomena salient to modern layouts, e.g., the impact of non-uniform interconnect stacks and partial routing obstacles on congestion. Second, the placement techniques that best control whitespace allocation in response to congestion (min-cut and annealing-based) can no longer efficiently handle the large number of movable objects present in modern designs. Third, incremental post-placement optimization *alone* is often insufficient as it cannot change the structure of global placement.

Challenges in congestion estimation [4]. A successful estimator must account for up to twelve metal layers with wire widths and spacings that differ by up to $20\times$. Blockages and per-layer routing rules must be modeled as well. Other constraints include via spacing rules and limits on intra-GCell routing congestion. After the 2007 and 2008 ISPD Routing Contests [51, 84], academic routers NTHU-Route 2.0 [14], NTUgr [47], FastRoute 4.0 [121], BFG-R [43] started to account for these issues. More recent routers — PGRIP [119], PGR (SGR) [77], GLADE [15, 70] — have improved solution quality and runtime, and account for different layer directives.

Routability-driven placement. In this context, several different optimization objectives can be pursued, such as ensuring 100% routability, even at the cost of significant routing runtime. Alternatively, placement solutions can be evaluated with a layer-aware global router with a short time-out, which nevertheless correlates with the final router (and is potentially based on the same software implementation). This intermediate objective is more amenable to optimizations in global placement because its quick evaluation facilitates a tight feedback loop. In other words, intermediate placements can be evaluated many times, allowing the global placer to make proper adjustments. Due to the correlation between the fast and the final routers' solutions, resulting routability-driven placements may fare better even with respect to the former, more traditional objective. This approach also facilitates early estimation of circuit delay and power in terms of specific route topologies. On the other hand, biasing the global placer away from its traditional optimization metrics to more sophisticated routability-based metrics (defined in Chapter II) may adversely affect the global placer's overall optimization capabilities.

1.2 Our Contributions

This dissertation develops the following contributions.

Standalone global routing based on integer-linear programming. As described in Chapter II, the global routing formulation involves an objective function subject to a set of constraints. This is reminiscent of linear programs (LP), and, if properly constructed, the obtained solution will be optimal (relative to its formulation). However, the traditional formulation is not scalable, even for small designs. To this end, in Chapter III, we present a scalable integer-linear program to optimally select a low-cost path for each net from a set of candidate paths. By controlling both (i) the number and (ii) the quality candidate paths, we are able to efficiently find high-quality solutions without incurring a high runtime overhead. In addition, our approach is *net-ordering* independent, as our linear program simultaneously routes all nets.

Standalone global routing based on Lagrangian relaxation. As stated in Chapter II, there are many efficient approaches and routing techniques to determine an optimal path for a given net. However, to satisfy all given constraints, the router often requires many iterations of ripping up nets in violation and finding better paths. The convergence problem is further exacerbated when the router cannot find better paths due to the non-changing landscape (e.g., persistent congestion). To this end, in Chapter IV, we present (i) routing framework that facilitates convergence by accounting for not only the current routes, but also the *history* of each net, and (ii) several generic techniques to improve the quality and performance of the router. By accounting for history, we ensure that the landscape is changing gradually, and relieve hard-to-route regions instead of moving them to different locations. Our individual techniques help control the cost-growths of each routing location, thereby preserving quality, and address performance bottlenecks, thereby improving runtime. Our implementation empirically validates the scalability of our algorithms. In addition to large publicly released benchmarks, we stress our system on a set of benchmarks on the order of those for trillion-gate systems (Section 4.6).

Simultaneous global placement and routing. To improve the quality of the placement solution, recent industrial practices have integrated global routers directly within the global placer in order to avoid future troublesome spots. Since the placer and router now iterate back and forth many times, the router must be fast as well as accurate. This congestion information directs the placer to regions where routing is difficult. However, the placer must take care to preserve the quality while improving routability. This is often done through the two general approaches of *whitespace injection* and *cell bloating*. However, the realization of these techniques are placer-specific. To this end, in Chapter V, we present a fully-integrated place-and-route framework that incorporates routability-driven components into a state-of-the-art global placer [66] and detailed placer [89]. In Chapter VI, we improved on the performance bottlenecks. To generate accurate congestion maps, we leverage the inherent interaction between the router and placer, and employ the Bellman-Ford algorithm to significantly improve routing runtime while preserving accuracy. We also identify the different types of congestion that is present during placement, and present several new techniques that efficiently addresses these difficult-to-route regions. Empirically, our implementation handles instances with millions of movable objects and nets without incurring large resource overhead.

Heterogeneous 3D technology. In addition to integration with other physical design tools, a global router can be used to evaluate routability for different technologies. In Chapter VII, we address the *buffer-explosion* problem, where the number of inserted buffers significantly increase with each technology node [97]. Here, we use two different technology nodes such that a significant number of buffers are housed on a separate, older technology die. We describe how we use a global router to (i) estimate routability on both dies (independently) and (ii) estimate the overall benefit of using two dies in the context of this heterogeneous 3D technology.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows. Part I provides the setting for our work. Chapter I presents the challenges of global routing. Chapter II formalizes the global routing problem, and outlines the relevant prior work in global routing. Part II covers our preliminary work in global routing. Chapter III describes a global router that routes all nets *simultaneously* using ILP, while Chapter IV describes a global router that routes all nets *iteratively* using history, e.g., negotiated-congestion. Chapter V describes our preliminary work for integrating a simplified global router into global placement to produce solutions such that the routing quality is improved. Part III extends the role of global placement to help facilitate resource management, both during the functional and physical design phases. Chapter VI improves upon our preliminary work on routability-driven placement by improving the scalability of congestion estimation and developing new techniques to relieve different types of congestion. Chapter VII addresses the *buffer-explosion* problem, discusses the benefits of moving buffers to a separate buffer die, and describes the usage of a global router within this design flow. Chapter VIII summarizes the thesis, and discusses topics for future research.

CHAPTER II

State-of-the-Art Global Routing Algorithms

In this chapter, we review the terminology and objectives of global routing, how it connects to detailed routing and global placement, and several known routing approaches.

2.1 Global Routing Terminology

A global routing instance is divided into two parts: the design's layout, and the design's netlist. The **design's layout** is represented as a three-dimensional $X \times Y \times Z$ routing grid G, where each $0 \leq z < Z$ represents a metal layer with dimensions $X \times Y$. Each layer consists of global routing cells or *GCells*, each with coordinate $g(x, y, z)$; the bottom left GCell of G to have coordinate $(0, 0, 0)$. To represent *preferred routing directions*, we limit the connectivity of GCells within each $X \times Y$ plane to be only *horizontal* or *vertical*. Therefore, each GCell with coordinate $g(x, y, z)$ is connected to four other GCells: two on the same plane, one leading to the layer above, and one leading to the layer below. To model routing resources, each edge e between two GCells g_i and g_j is assigned a *routing capacity* $cap(e)$, defined as the number of times e *can be* prescribed. Similarly, each edge e also has a *routing usage* $usage(e)$, which is defined as the number of times e *has been* prescribed. In this model, we distinguish the edges that connect GCells in the same layer as *routing segments*, and edges that connect GCells across different layers as *vias*; as

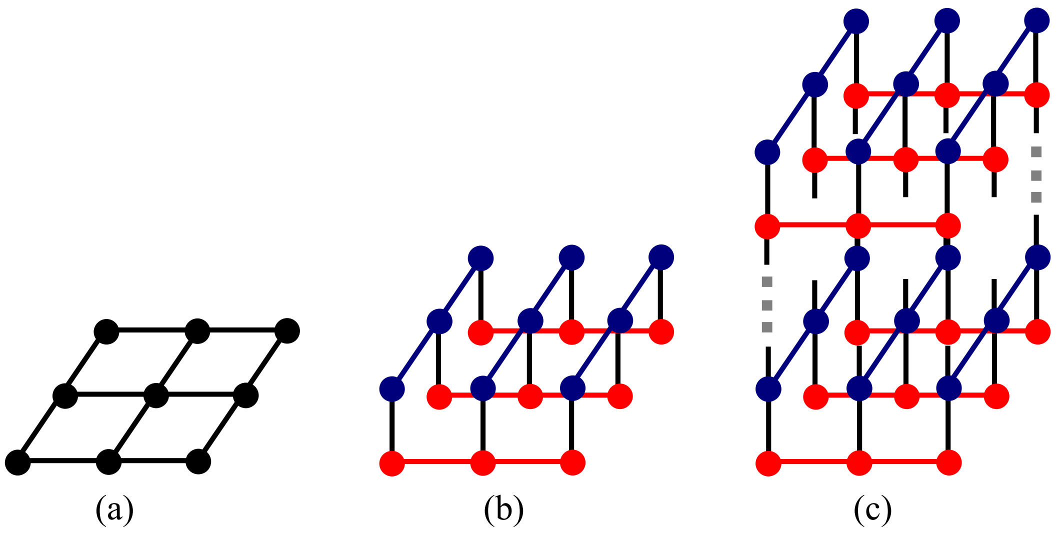


Figure 2.1: The global routing grid formats. (a) A two-dimensional grid, where horizontal and vertical tracks are on the same layer. (b) A 2.5-d grid, with one layer of horizontal tracks (red), one layer of vertical tracks (blue), and a layer of connecting vias (black). (c) A three-dimensional grid, with alternating horizontal and vertical routing layers connected by vias.

routing layers and vias are made up of different materials, e.g., copper and tungsten, vias are sometimes considered less desirable than routing segments. The full routing grid abstraction is illustrated in Figure 2.1. Typically, the routing grid is *two-dimensional*, where horizontal and vertical tracks are on the same plane. At older technology nodes, the grid was limited to two metal layers. At newer technology nodes, the number of metal layers have been increased to upwards of ten or more, where horizontal and vertical layers alternate. To improve scalability, global routers have collapsed the *three-dimensional* routing grid to a 2.5-d routing grid, where all horizontal tracks are in one layer, all vertical tracks are in the other layer, and the two layers are connected by vias.

The **design's netlist** is comprised of *nets*, where each net consists of a set of gates or cells that must be connected. To represent nets on the routing grid, each cell's location is snapped to the closest GCell location. A net is *routed* if the set of GCells is connected by a set of edges in the routing grid (Figure 2.2).

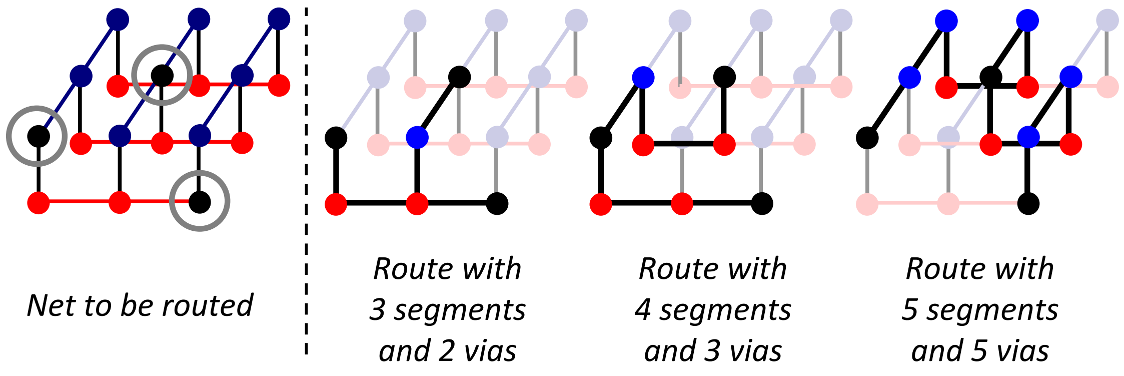

Net to be routed
Route with 3 segments and 2 vias
Route with 4 segments and 3 vias
Route with 5 segments and 5 vias

Figure 2.2: An example of a net that requires a route on a 2.5-d routing grid (left), where the three circled points need to be connected by a combination of routing segments and vias. The three on the right depict several possible routes, each using a different number of edges.

The **design's quality** is commonly measured by some combination of its (i) (weighted) wirelength, (ii) overflow, and (iii) congestion. We define the *weighted wirelength* of a net n in the netlist N as the weighted sum of its routing segments and vias

$$wirelength(n) = \alpha \times segments(n) + \beta \times vias(n) \tag{2.1}$$

where $segments(n)$ is the number of horizontal and vertical segments of n, $vias(n)$ is the number of vias of n, and α and β represent the relative importance of routing segments and vias. As traversing from one metal layer to another is becoming costly, vias have non-trivial timing effects and they may block several routing tracks [106]. Therefore, vias can have higher priority than routing segments [51]. For each edge e in the routing grid, we define the *overflow* of e as the difference between the edge's usage and capacity if the usage exceeds the capacity, and zero otherwise.

$$OF(e) = \max(0, usage(e) - cap(e)) \tag{2.2}$$

Similarly, we define the *congestion* of e as the ratio between the edge's usage and capacity

$$C(e) = \frac{usage(e)}{cap(e)} \tag{2.3}$$

The quality of the netlist N is measured by its *weighted total wirelength,*

$$\sum_{n \in N} wirelength(n) \tag{2.4}$$

its *total overflow*, defined as the sum of all edge overflows in each net,

$$TOF(N) = \sum_{e \in E} OF(e) \tag{2.5}$$

and its *maximum overflow*, defined as the maximum of all edge overflows.

$$MOF(N) = \max_{e \in E} OF(e) \tag{2.6}$$

Here, E is defined as the set of edges of the routing grid G.

2.2 Global Routing Formulation and Objectives

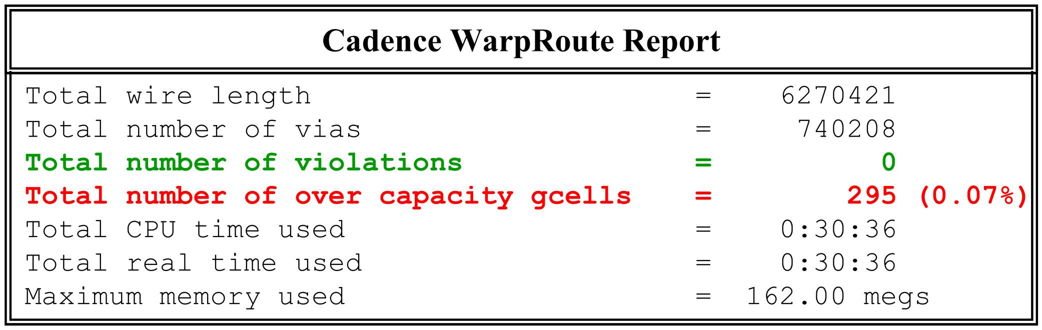
Traditionally, the only objective for global routing is to minimize total wirelength given that the solution is *legal*, i.e., where the usage of each edge does not exceed its capacity.

$$\min \sum_{n \in N} length(n) \quad s.t. \quad usage(e) \leq cap(e)\; \forall e \in E \tag{2.7}$$

Here, N is the set of all nets, $usage(e)$ and $cap(e)$ are the respective usage and capacity of edge e, and E is the set of all edges in the routing grid. However, modern global routers must be able to handle millions of objects, account for different technology constraints, and optimize for multiple objectives, all while maintaining a reasonable runtime.

Routing violations and wirelength. Typically, the number of violations should be zero, i.e., $MOF(N) = TOF(N) = 0$, but a purely legal global routing solution is not required. As illustrated in Figure 2.3, an excerpt from a *Cadence WarpRoute* report on a test benchmark shows that although global routing reported 295 GCells with violations, the detailed routing solution is legal.[1] As long as the percentage of violations is small, detailed routing is usually able to compensate.

[1] In this chapter, we limit our discussion to edge-centric violations, and include GCell-centric discussion in Chapters V and VI. In general, there is no commonly-agreed GCell-centric violation definition.



Cadence WarpRoute Report

```
Total wire length                          =    6270421
Total number of vias                       =     740208
Total number of violations                 =          0
Total number of over capacity gcells       =        295  (0.07%)
Total CPU time used                        =    0:30:36
Total real time used                       =    0:30:36
Maximum memory used                        =  162.00 megs
```

Figure 2.3: Excerpt from Cadence WarpRoute on a test benchmark. Notice that although global routing produced a total of 295 GCells with violations, the final result given by detailed routing has none. This is typical for industry circuits.

Modeling technology constraints. With older technology nodes, there were only two routing layers, where a net's routing segment cost a unit length (e.g., one routing edge). However, at lower technology nodes and increased metal layers, new constraints include: (i) different wire widths and spacings, (ii) routing blockages, and (iii) net pins on different metal layers. These will be discussed in further detail in Chapter V.

2.3 Previous Approaches in Global Routing

In this section, we outline the previous approaches of (i) single-net routing, (ii) standalone global routing frameworks, and (ii) incorporating global routing in placement.

2.3.1 Prior Work in Routing (Point-to-Point) Single Nets

Techniques to construct an optimal path from a single source to a single target[2] are well-known, and represented by Dijkstra's Algorithm and A*-search [30]. While these methods enable maximum flexibility, they often incur a runtime penalty. This section summarizes the different common approaches used to generate a (possibly suboptimal) route; the following section discusses how global routing frameworks leverage these point-to-point methods.

[2]This can be generalized to multiple sources and targets.

Pattern routing. Using simple and predetermined routes, pattern routing significantly reduces the problem's solution space. Instead of having restrictions placed on each routing segment, each net is limited to a small number of shapes. A two-pin net is commonly mapped to an L shape, where only one bend is used and the wirelength is optimal, or a Z shape, where two horizontal segments are connected with a middle vertical segment or vice versa. Kastner et al. [63] have shown that in standard application specific integrated circuits (ASICs), pattern routing is efficient, as it minimizes via count and increases scalability. Further work done by Westra et al. [117] shows that the majority of two-pin nets can be routed using L shapes. Typically, pattern routing chooses from a collection of finite routing topologies, and is more flexible than using only Ls and Zs.

Monotonic routing. In monotonic routing, the search direction is only allowed up and to the right. That is, edges that lead down or to the left (e.g., detoured) are forbidden. Monotonic routing is often implemented using dynamic programming (Algorithm 1). Lines 4-11 initialize the costs located on the borders. Lines 12-26 then propagate the costs at the border in a topological manner (towards the target) such that the optimal cost at (i, j) is only dependent on costs at locations $(i - 1, j)$ and $(i - 1, j)$. Line 27 (Algorithm 2) records the route by backtracking from the target.

Maze routing. The most versatile routing technique, maze routing uses shortest-path algorithms such as Dijkstra's Algorithm and A*-search [30, Section 24.3] to connect terminals along the routing grid. While optimal paths can be found for pairs of terminals, the order in which nets are routed has a profound effect on solution quality and routed length. As a result, maze routing must be applied many times with heuristic net orderings to find legal solutions. Moreover, vias are modeled *explicitly* to prevent unnecessary detouring.

Algorithm 1 Monotonic Routing.

Input: Net n
Output: route $n.route$

```
1: $ll = n.lowerLeftCoordinate$;
2: $ur = n.upperRightCoordinate$;
3: $cost[ll.x][ll.y] = 0$;
4: for $i$ from $ll.x + 1 \to ur.x$ do
5:     $cost[i][ll.y]$ = COST($(i-1, ll.y) \sim (i, ll.y)$) $+cost[i-1][ll.y]$;
6:     $parent[i][ll.y]$ = $(i-1, ll.y)$;
7: end for
8: for $j$ from $ll.y + 1 \to ur.y$ do
9:     $cost[ll.x][j]$ = COST($(ll.x, j-1) \sim (i, ll.y)$) $+cost[ll.x][j-1]$;
10:     $parent[ll.x][j]$ = $(ll.x, j-1)$;
11: end for
12: for $i$ from $ll.x + 1 \to ur.x$ do
13:     for $j$ from $ll.y + 1 \to ur.y$ do
14:         $leftEdge$ = $(i-1, j) \sim (i, j)$;
15:         $leftCost$ = COST($leftEdge$) $+cost[i-1][j]$;
16:         $downEdge$ = $(i, j-1) \sim (i, j)$;
17:         $downCost$ = COST($downEdge$) $+cost[i][j-1]$;
18:         if $downCost < leftCost$ then
19:             $cost[i][j] = downCost$;
20:             $parent[i][j] = (i, j-1)$;
21:         else
22:             $cost[i][j] = leftCost$;
23:             $parent[i][j] = (i-1, j)$;
24:         end if
25:     end for
26: end for
27: TRACE_PATH($n$);
```

Algorithm 2 Path-tracing Algorithm. TRACE_PATH

Input: Net n
Output: $n.route$

```
1: $cur$ = $n.target$;
2: while $cur$ != $n.source$ do
3:     $par = parent[cur]$;
4:     ADD_EDGE($n.route$, $(par, cur)$);
5:     $cur$ = $par$;
6: end while
```

2.3.2 Prior Work in Standalone Global Routers

Using point-to-point techniques described in the previous section, global routers (iteratively) construct paths for every net such that all constraints are satisfied. This section outlines several global-routing frameworks, including those based in satisfiability (SAT) and linear programming (LP), and those based on Lagrangian relaxation.

SAT- and ILP-based routing. By modeling routing constraints by Boolean formulas in CNF, Nam et al. [83] developed a SAT-based detail router which routes all nets simultaneously. Using ILP, this formulation can be extended to route as many nets as possible [120]. ILP-based routing has traditionally been avoided due to its lack of scalability. An early attempt by Burstein and Pelavin [10] could not be efficiently implemented because ILP solvers were not sufficiently powerful. However, after major improvements in ILP solvers, the idea of routing optimally using ILP became viable. M. Cho and D. Pan developed BoxRouter 1.0 [20]. After decomposing multi-pin nets into two-pins subnets, BoxRouter 1.0 uses pattern routing and begins at the most congested region. Starting within a small bounding box, it optimally routes as many nets in the region as possible using only L patterns; the remaining unrouted nets are given to a maze router. The bounding box is iteratively expanded using a *progressive* ILP formulation that extends partially-routed nets with additional L-shaped segments. Then maze routing is invoked to complete nets that did not route. Such steps are repeated until the entire global routing grid is subsumed. Given that ILPs are solved optimally, using powerful ILP solvers can only improve runtime. However, a faster ILP solver may facilitate a more comprehensive ILP formulation.

One common method used to improve the scalability of ILP-based routing techniques is to relax the ILP problem into an easier linear programming (LP) problem. Multicommodity flow (MCF) based routers take this approach [2, 41]. An approximation technique incrementally adjusts routing edge weights and builds new Steiner tree topologies for each net at every iteration to solve the LP. BoxRouter 1.0 has been compared to a recent MCF-based router and was found to be superior in speed and solution quality [20]. More recently, the culmination of these techniques were implemented in CGRIP [100], where the design is first divided into many small regions, and then each region is routed (solved) simultaneously using their ILP formulation. The regions are then reintegrated to account for nets that cross multiple regions. The original CGRIP used a large number of processors, e.g., one for each window; a more scalable version, coalesCgrip, was introduced later during the ISPD 2011 Routability-driven Contest [108].

History-based routing. Instead of being limited to locally temporal metrics such as current congestion, routers that employ *history* maintain routing-violation information from previous iterations, and incorporate that data into the cost function. This technique is founded on *Lagrangian relaxation*, which was popularized by PathFinder [80]. Because the original formulation (Equation 2.7) is too difficult to solve without violating any constraints, we relax the constraints into penalty functions, and incorporate them into the objective function, we define the *cost* of a routing solution as

$$\sum_{n \in N} cost(n) \tag{2.8}$$

where the cost of a net n is defined as

$$cost(n) = \sum_{e \in n} cost(e) \tag{2.9}$$

Here, the cost of an edge e encapsulates the specific problem instances. Typically, the cost is based on (but not limited to) e's usage, layer, congestion, or other technology-dependent parameters. For simplicity, we limit our discussion to PathFinder's edge cost formulation, but other parameters can be easily incorporated.

$$cost(e) = base(e) + \lambda(e) \times penalty(e) \tag{2.10}$$

Here, $base(e)$ is the edge's base (e.g., unit) cost, $\lambda(e)$ is the edge's Lagrangian multiplier, and $penalty(e)$ represents the current penalty (e.g., congestion) on e. In this formulation, $\lambda(e)$ represents the edge's history, and encapsulates the frequency at which the edge has violations. To ensure convergence, this value monotonically increases. If the solution has overflow (or congestion), then the Lagrangian multiplier will increase, thereby increasing the cost of the total solution. Therefore, the goal is to minimize the total cost of all nets. Notice, however, if the solution contains no violations, the cost will be no different than the original formulation, and we have found a solution to the non-relaxed problem.

From the ISPD 2007 and 2008 Global Routing Contests [51, 84], the vast majority of successful academic routers have employed the use of history, such as FGR [93], Archer [86], NTHU-Route 2.0 [14] and NTUgr [47].

2.3.3 Using Global Routing Estimates in Placement

With increasing design complexity, optimizing traditional placement metrics is insufficient for successful routing [4, 94]. To mitigate routing failures, *routability-driven placers* incorporate route estimation as part of their flow. In this context, the placer is given information about difficult-to-route areas, often in the form of *congestion maps*, where edge-centric routing congestion is represented by GCell-centric congestion. This is typically done in two methods: (i) using congestion-estimation techniques, and (ii) using global routing techniques to estimate congestion. Previously-developed routers include work from Hadsell and Madden (Fengshui with Chi dispersion) [36], M. Cho and D. Pan (BoxRouter 1.0) [20], Roy and Markov (FGR 1.0) [93], as well as M. Pan and C. C. N. Chu (FastRoute) [87, 88]. Fengshui, BoxRouter 1.0, and FGR 1.0 minimize total routed wirelength, while FastRoute minimizes its runtime at the cost of higher wirelength.

APPROACH	TECHNIQUE
STATIC	Rent's Rule [34, 35, 79, 124]
	net bounding box [11, 58]
	Steiner trees [94]
	pin density [9, 128]
	counting nets in regions [114]
PROBABILISTIC	uniform wire density [38, 48, 102]
	pseudo-constructive wirelength [61]
	smoothened wire density [105]
	pattern routing [117]
CONSTRUCTIVE	using A*-search [118]
	using a global router: • FastRoute [121] in IPR [24] • BFG-R [43] in SimPLR [65]

Table 2.1: Previous congestion estimation for placement.

Congestion maps indicate regions where routing will be difficult, and are used to guide optimization during placement. They are generated using: (i) static approaches, where the congestion map is fixed for a placement instance, (ii) probabilistic approaches, where net topologies are not fixed, and probabilistically determined, and (iii) constructive approaches, where a simplified global router generates approximate net routes. Traditionally, the first two options have been the most popular, but the last option has recently been gaining acceptance thanks to advanced global routers designed to handle greater layout complexity. Empirical evidence from the ISPD 2011 Routability-driven Contest [108] suggests that both *probabilistic* and *constructive* methods are viable and scalable. Table 2.1 summarizes these approaches. During the ISPD 2011 Routability-driven Contest [108], SimPLR integrated the global router BFG-R [43], whereas Ripple [38] and NTUPlace4 [48] adopted probabilistic congestion estimation [102].

Placement optimizations are applied throughout the entire placement flow: (i) during global placement, (ii) modifying intermediate solutions, (iii) during legalization and detailed placement, and (iv) as a post-placement processing step (Table 2.2). In global placers, the most popular techniques are *cell bloating* and *whitespace injection*. Depending on the placer type, e.g., quadratic and min-cut, the implementation of these techniques will require placer modification, including changing the optimization function. In detailed placers, the most popular techniques are *cell swapping* and *cell shifting*. Additional optimizations can be applied to intermediate (or near-final) placement solutions, and then passed on to the next step of the design flow. During the ISPD 2011 Routability-driven Contest [108], both SimPLR [65] and Ripple [38] used congestion maps to *bloat cells* and modify the anchor positions during quadratic placement. NTUPlace4 [48] used congestion maps when modeling pin density.

PLACEMENT PHASE	TECHNIQUE
GLOBAL PLACEMENT	relocating movable objects: • moving nets [38, 58] • modifying forces [26, 102] • incorporating congestion in objective function [48, 105] • adjusting target density [65]
	cell bloating [9, 38, 39, 65]
	macro porosity [48, 58]
	pin density control [48]
	expanding/shrinking placement regions [91]
INTERMEDIATE	local placement refinement [24]
LEGALIZATION AND DETAILED PLACEMENT	linear placement in small windows [54, 94]
	congestion embedded in objective function [126]
	cell swapping [24, 38, 65]
	cell shifting [33, 48]
POST PLACEMENT	whitespace injection or reallocation [75, 94, 123]
	simulated annealing [18, 40, 112]
	linear programming [76]
	network flows [113, 115]
	shifting modules by expanding GCells [126]
	cell bloating [95]

Table 2.2: Prior congestion-driven placement techniques.

PART II

Global Routing in the Context of High-performance Design Flow

CHAPTER III

Sidewinder: A Scalable ILP-based Router

In this chapter, we develop a global router that incorporates an ILP formulation that (i) allows the router to have flexibility when routing nets, and (ii) is scalable (and adaptable) for larger designs. First, we determine a number of different routes for each net. Second, we select the top two candidates for each net based on the current congestion. Third, we create a scalable ILP formulation that lets the ILP solver choose the better route candidate. As shown in Section 3.3, empirically, on the ISPD98 benchmarks [50], this formulation alone routes 98$\%$ of nets with optimal wirelength and minimal via count, but remaining nets require small detours.

3.1 Introduction

The first ILP-based router was proposed by Burstein and Pelavin [10] but was impractical because ILP solvers of the day were unacceptably slow. ILP solvers have improved dramatically in terms of speed and efficiency in the past twenty years and, M. Cho and D. Pan [20] have successfully implemented an ILP-based router BoxRouter 1.0 with pre- and post-processing to simplify the problem. Like BoxRouter 1.0, we consider routing optimally all two-pin nets with L shapes first. However, instead of iteratively expanding a small bounding box, we consider the entire routing grid during each pass. In addition to L

shapes, we also consider all Z shapes and selected C shapes (Figure 3.2).

Sidewinder is much simpler than existing routers because the majority of work is done by the ILP solver. Unlike the ILP formulations used in BoxRouter 1.0 [20], Sidewinder's pattern routes allow at most three bends per two-pin subnet and detours of at most four GCells in length. With these restrictions relaxed, any remaining nets can be routed with a simple post-routing step in all the designs we considered. On the other hand, Figure 2.3 suggests that Sidewinder is already a viable global router because post-processing can be performed by existing detail routers. Sidewinder's ILP formulation can also be used in the BoxRouter flow to improve via count and detours.

The following key ideas are proposed in this work:

- selection of two least congested patterns per net.
- search over all two-bend Z-shaped routes.
- use of detoured two-bend and three-bend C-shape routes.
- congestion-based ILP formulation.
- congestion map updates between ILP calls.
- an incremental ILP for all nets that is guaranteed to never make solutions worse.

The rest of this chapter is structured as follows: Section 3.2 describes the problem formulation in detail. Section 3.3 has the experimental setup and results. Section 3.4 concludes this chapter and mentions future work.

3.2 Sidewinder

We present Sidewinder's high-level flow, related algorithms, and the ILP formulation.

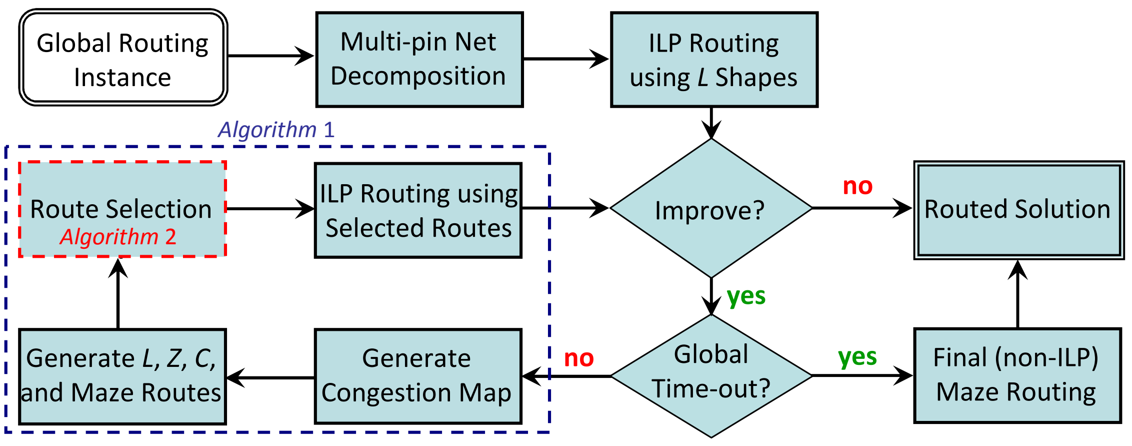

Global Routing Instance
Multi-pin Net Decomposition
ILP Routing using L Shapes
Algorithm 1
Route Selection
Algorithm 2
ILP Routing using Selected Routes
Improve?
no
Routed Solution
yes
Generate L, Z, C, and Maze Routes
Generate Congestion Map
no
Global Time-out?
yes
Final (non-ILP) Maze Routing

Figure 3.1: High-level flow of Sidewinder. We first create an initial solution using only L shapes. Next, we build a congestion map based on the current solution to use as a guide for the new solution. For net route candidates, we consider Ls, Zs, Cs, and a maze route. Once all nets are processed, an ILP is formed and solved. This cycle continues until the new solution has the same cost as the current solution. Once there is no more improvement, maze routing is applied to yield the final routing solution.

3.2.1 High-level Framework

We only consider the routing of two-pin nets; multi-pin nets are decomposed into multiple two-pin nets. The terminals of each net are located within their respective GCells. As shown in Figure 3.1, we first generate an initial routing solution using only L shapes (initial routing). Using this current solution, we build a congestion map to guide the routing of the new solution. For each net, we consider Ls, Zs, Cs, and a maze route as possible route candidates. This specific portion is discussed in greater detail in Algorithm 3, Algorithm 4, and Section 3.2.2. After all the route candidates have been selected, we formulate this problem into an ILP and generate the new routing solution. If this new solution is better (higher objective function) than the previous solution, this process is repeated. Once there is no more improvement, we apply a pass of maze routing to route all outstanding nets.

3.2.2 Algorithm Design

The iterative portion of Sidewinder is given in Algorithm 3. The first iteration routes as many subnets as possible using Ls. In subsequent iterations, alternative route types of Ls, Zs, Cs, and *maze* are evaluated using a congestion map. Line 5 constructs a congestion map based on the current routing solution S. Lines 13-29 generates all route candidates for each net. To improve routability, we evaluate all unrouted nets before routed nets. Lines 13-20 selects the top num_routes candidates for all currently-routed nets, with one choice being the current route. Similarly, lines 21-29 selects the top num_routes candidates for each currently-unrouted net. Line 30 solves the ILP formulation, and lines 31-36 evaluates the solution progression.

For each net, we only consider legal route candidates, e.g., detoured routes that are not within the routing grid are not allowed. Each of the shapes are also considered “sufficiently different” – this gives the router more flexibility and freedom. We emphasize that the two chosen routes are always different. In the case where the maze route is a duplicate pattern route, the maze route is removed and the next best route comes off the priority queue.

Once the two routes are selected, the congestion map is updated. If the net was routed, the current route is given a weight of 0.9 and the new candidate 0.1. If the net was not routed, each candidate is given a weight of 0.5. Notice that the congestion map is updated after each net has been processed. This guides the router such that the new route choices will not create new congestion areas.

After each net has two possible route candidates, we create the ILP formulation and solve. This yields a new routing solution S'. If the solution quality of S' is better than the solution quality of S, then we set $S = S'$, and the process is repeated. From our formulation, we define the quality of a routing solution to be the objective function returned by the ILP solver. A higher objective value implies more nets have been routed. Once the

Algorithm 3 High-level Iterative Algorithm of Sidewinder.

Input: Routing Grid G, Netlist N, Route Types RT,
Number of considered routes *num_routes*, (Partially) Routed Solution *S*

Output: New Routed Solution *S'*

```
1: $improve$ = **true**
2: $nets\_unrouted = \varnothing$;
3: $nets\_routed = \varnothing$;
4: **while** $improve$ **do**
5:     $CM$ = GENERATE_CONGESTION_MAP($G$, $S$);
6:     **for** all nets $n \in N$ **do**
7:         **if** IS_NET_ROUTED($n$, $S$) **then**
8:             ADD_TO_LIST($nets\_routed$, $n$);
9:         **else**
10:            ADD_TO_LIST($nets\_unrouted$, $n$);
11:        **end if**
12:    **end for**
13:    **for** all nets $n \in nets\_unrouted$ **do**
14:        **for** all route types $rt \in RT$ **do**
15:            $pq$.INSERT(GENERATE_ROUTE($n$, $pt$, $CM$));
16:        **end for**
17:        **for** $i = 0 \rightarrow$ $num\_routes$-1 **do**
18:            $routes[n][i] = pq$.POP();
19:        **end for**
20:    **end for**
21:    **for** all nets $n \in nets\_routed$ **do**
22:        **for** all route types $rt \in RT$ **do**
23:            $pq$.INSERT(GENERATE_ROUTE($n$, $pt$, $CM$));
24:        **end for**
25:        $routes[n][0]$ = CURRENT_ROUTE($n$, $S$);
26:        **for** $i = 1 \rightarrow$ $num\_routes$-1 **do**
27:            $routes[n][i] = pq$.POP();
28:        **end for**
29:    **end for**
30:    $S'$ = SOLVE_ILP(GENERATE_ILP_FORMULATION($G$, $N$, $routes$));
31:    $improve$ = OBJECTIVE_VALUE($S'$) > OBJECTIVE_VALUE($S$);
32:    **if** $improve$ **then**
33:        $S = S'$;
34:    **else**
35:        $S' = S$;
36:    **end if**
37: **end while**
```

objective value stabilizes, i.e., OBJECTIVE_VALUE(S) $=$ OBJECTIVE_VALUE(S'), this iterative portion terminates.

Algorithm 4 Route Selection.

```
Input:    Routing Grid $G$, Route $r$
Output:   Minimum Number of Free Segments Along the Path segs_free_min,
          Total Number of Free Segments Along the Path segs_free_total

1:  $segs\_free\_min = 0$;
2:  $segs\_free\_total = 0$;
3:  if IS_ROUTE_ILLEGAL($G$, $r$) then
4:      $segs\_free\_min$ = route_illegal;
5:      $segs\_free\_total$ = route_illegal;
6:  end if
7:  for all edges $e \in r$ do
8:      if $G[e].capacity < 0$ then
9:          if $segs\_free\_min \geq 0$ then
10:             $segs\_free\_min =$ -1;
11:         else
12:             - -$segs\_free\_min$;
13:         end if
14:     else
15:         $segs\_free\_min$ = MIN($segs\_free\_min$, $G[e].capacity$);
16:         $segs\_free\_total$ += $G[e].capacity$;
17:     end if
18: end for
```

The algorithm for route calculation and selection is given in Algorithm 4. Each candidate route is given two metrics: minimum number of free segments ($segs\_free\_min$) and total number of free segments ($segs\_free\_total$). $segs\_free\_min$ is found by taking the minimum available space/segment for each segment in the route. If a segment has no room (capacity = 0) or is overfilled (capacity $<$ 0), the priority is the -(total number of routing violations). In other words, routes with overflow have a negative priority (less desirable) while routes without any violations have a positive priority (more desirable). Likewise, $segs\_free\_total$ is found by summing up the total number of free space across the route.

Once all the route priorities are calculated, they are ranked by $segs\_free\_min$. That

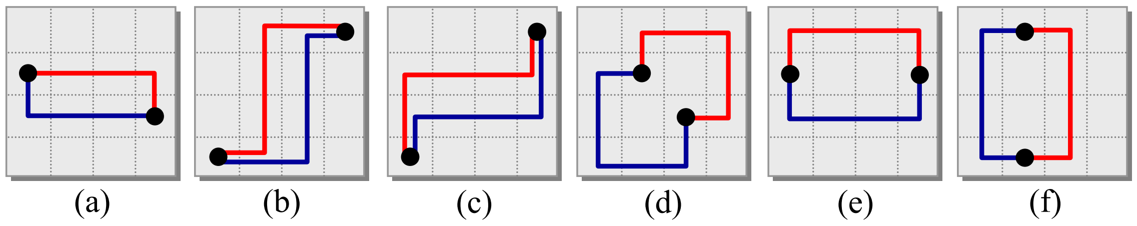


Figure 3.2: Patterns Sidewinder considers when choosing routes. (a) Two different L shapes, (b) All possible vertical Zs, (c) All possible horizontal Zs, (d) C shapes – detouring one unit in the vertical direction, (e) C shapes – detouring one unit in the horizontal direction, (f) C shapes – detouring one unit in both the horizontal and vertical direction.

is, the least congested routes are the top choices. $segs\_free\_total$ is only used in case of a tie between routes that have the same $segs\_free\_min$. Thus, the most desirable route is the one with the most total available capacity. Note that with this formulation, there are ALWAYS at least two legal and "sufficiently different" routes available. With this formulation, we *guarantee that the ILP solution will be no worse than the previous.* Each subsequent ILP instance routes at least as many nets as the current ILP instance. In the worse case, the same nets will be routed, causing the objective function to stay constant.

3.2.3 ILP Formulation

In this section, we present the general ILP formulation (Algorithm 5) that considers k types of routes. In our implementation, we let $k = 2$, and in the first ILP iteration, we only consider L-shaped routes.

Recall that we choose two possible route candidates for each net n in the netlist N. In the ILP, this is represented with two 0-1 variables, x_{n_1} and x_{n_2}. A value of 0 represents the route was not chosen; the value of 1 represents the route chosen for the net. The first three constraints guarantees that at most one route out of the two will be selected (either one route will be chosen or no routes will be chosen). The fourth constraint states that for all $North$ routing edges $g(x, y) \sim g(x, y + 1) \in G$, the summation of all selected routes must be less than or equal to $cap(g(x, y) \sim g(x, y + 1))$, the total capacity of $g(x, y) \sim g(x, y+1)$. That is, the sum of routing segments assigned through a GCell must be less than or equal to the total capacity of the edge. Similarly, the next three constraints ensure that $South$, $East$, and $West$ edge capacities are respected. Note that only the $North$ and $East$ (or some similar variation) constraints are needed, as the $North$ and $South$ constraints are the same and the $East$ and $West$ are the same.

Algorithm 5 Sidewinder's ILP Formulation.

Inputs		
G	:	routing grid
$X \times Y$	:	width X and height Y of G
$cap(g(x, y) \sim g(x + 1, y))$	:	capacity of horizontal edge $g(x, y) \sim g(x + 1, y)$, where $0 \leq x < X - 1$ and $0 \leq y < Y$
$cap(g(x, y) \sim g(x, y + 1))$	:	capacity of vertical edge $g(x, y) \sim g(x, y + 1)$, where $0 \leq x < X$ and $0 \leq y < Y - 1$
N	:	netlist

Variables		
$x_{n_1}, \ldots, x_{n_k}$	:	k Boolean route variables for each net $n \in N$
$w_{n_1}, \ldots, w_{n_k}$	:	k net (real) weights, one for each net $n \in N$

Maximize:

$$\sum_{n \in N} w_{n_1} \cdot x_{n_1} + \cdots + w_{n_k} \cdot x_{n_k}$$

Subject to	
$x_{n_1} + \cdots + x_{n_k} \leq 1$	$\forall n \in N$
$x_{n_1}, \ldots, x_{n_k} \in [0, 1]$	$\forall n \in N$
$\sum_{n \in N} x_{n_1} + \cdots + x_{n_k}$	$\forall n_k$ that use edge $g(x, y) \sim g(x + 1, y)$
$\leq cap(g(x, y) \sim g(x + 1, y))$	$0 \leq x < X - 1, 0 \leq y < Y$
$\sum_{n \in N} x_{n_1} + \cdots + x_{n_k}$	$\forall n_k$ that use edge $g(x, y) \sim g(x, y + 1)$
$\leq cap(g(x, y) \sim g(x, y + 1))$	$0 \leq x < X, 0 \leq y < Y - 1$

The next variables w_{n_1} and w_{n_2} are the corresponding weights given to each route. These weights are determined by the type of route x_{n_1} and x_{n_2} are. Strictly speaking, a route with a higher coefficient is more preferred than a route with a lower coefficient. Since we consider a number of routes with different wirelength and bends (an L has less wirelength and fewer bends than a detour), we assign different weights to the objective function based on the type of route selected. Since the objective function is maximized, we value Ls the most, followed by Zs, then Cs, and then maze routes. Note that although we consider many different routes, the number of variables needed is still only two per subnet, ensuring the scalability of our ILP formulation.

3.2.4 Insights

During our preliminary work, we have evaluated a number of different ILP formulations to global routing. We quickly observed that all formulations that scale to a large number of nets fell into the category of pattern routing. That is, they would only allow a small number of configurations per net. Furthermore, ILP formulations with only two patterns per net were solved an order of magnitude faster than those with four or more patterns per net.

While our observations about efficient ILP formulations are consistent with the success of L-shape routing in BoxRouter 1.0, the choice of L-shapes is not as critical. Thus our first insight is as follows: *Select routing patterns other than L-shapes for nets and allow for dynamic selection of pattern shapes.*

For further studies, we extracted several small but difficult routing instances from common benchmarks. In some of the instances, only about half the nets could be routed with Ls due to capacity constraints. We have evaluated several simple patterns, including Z-shapes where the middle segment would cross the midpoint of the net's bounding box.

We found that allowing this pattern provides only marginal (if any) improvement to L-only ILPs. However, *including shapes with slight detours (which we term as C-shapes) allowed us to route significantly more nets.*

Our third insight is *routes should be evaluated based on congestion, rather than on length or via count, to determine the best candidates.* For the initial ILP formulation, we select the two best routes based on congestion if the net was not routed previously and the current and best routes if the net was routed. We noticed that the runtime of the ILP solver decreased dramatically the more accurate we were at predicting the possible routes.

Our final insight is that *all Z-shaped routes should be considered rather than only ones that cross the midpoint of a nets' bounding box.* For a given net, we can scan the congestion map and find quickly the least congested Z-shaped routes. We noticed that this new flexibility noticeably improved our solution quality.

3.2.5 Sidewinder vs. BoxRouter 1.0

Comparing our ILP formulation with BoxRouter 1.0 — the only scalable ILP router in the literature — we note several important differences:

- BoxRouter's ILP is applied to a small region and includes only L-shaped routes; our formulation is applied to the entire global routing grid and after the first iteration also includes all possible C-shapes and Z-shapes.
- For long nets, BoxRouter's ILP routes one portion of the net at a time, whereas Sidewinder's ILP routes entire nets in all cases.
- At each iteration, BoxRouter's *progressive* ILP extends its *current region* to a slightly larger region and extends nets present in both regions by new L-shaped segments. Therefore, long nets may be routed with two bends per region,[1] whereas Sidewinder's

[1]Except in cases where the L is degenerate — a flat wire

formulation is *global* and does not allow more than three bends per subnet.

- BoxRouter's ILP formulation is not sensitive to congestion, but is formulated for the most congested region in its first iteration. In contrast, Sidewinder's ILP formulation is global. The second iteration (and beyond) explicitly accounts for congestion when selecting two patterns for each net. Moreover, the status of the internal congestion map is dynamically updated during the ILP construction.

3.3 Empirical Validation

We implemented Sidewinder as follows. The high-level algorithms are written in C++; we used CPLEX v.10.1 [31] as our ILP solver. Using FLUTE [25], we decompose all multi-pin nets into two-pin subnets. For our ILP cost function, we use the following pricing scheme for the different patterns: 1.00 for Ls, 0.99 for Zs, 0.98 for Cs, and 0.97 for the maze route. Note that this formulation directly accounts for both bends (vias) and wirelength. L-shapes are the most preferred route, as they have the fewest number of bends – zero or one. After Ls, Z-shapes are the most preferred, as they have the same (minimal) wirelength and only one extra bend. Next, C-shapes have an additional two units of wirelength and one additional bend. When no pattern route is legal, a maze route used as the last choice. In practice, the maze routes have more bends and wirelength than any of the other patterns. The chosen coefficients both encourage the use of short (L-shapes) routes as well as enable a degree of flexibility for detours. All experiments were performed on an AMD Opteron 2.4 GHz machine with 4 GB of memory.

Routability results for Sidewinder on the ISPD98 benchmarks [50] are shown in Table 3.1. We list the percentage of nets routed by Sidewinder, the number of iterations necessary and the total runtime for each benchmark. The ILP portion of Sidewinder is successful in routing 99.86% of all nets. Note that 100% routability is not required - the

Benchmark	Size ($X \times Y$)	Total Nets	Total Routed	# ILP Iters.	Runtime (min)
IBM01	64×64	11507	99.36%	12	231
IBM02	80×64	18429	99.95%	8	92
IBM03	80×64	21621	99.99%	6	93
IBM04	96×64	26163	99.50%	6	217
IBM05	128×64	27777	100%	1	< 1
IBM06	128×64	33354	99.98%	6	130
IBM07	192×64	44394	99.94%	6	100
IBM08	192×64	47944	99.98%	6	120
IBM09	256×64	50393	99.99%	6	277
IBM10	256×64	64227	99.98%	5	103
Average			99.86%		

Table 3.1: Results of routability for Sidewinder on the ISPD98 benchmark suite [50] BEFORE FINAL ROUTING.

ISPD98	BoxRouter 1.0			FGR 1.0			Sidewinder		
Benchmarks	Over-fllow	Via Count	Routed Length	Over-fllow	Via Count	Routed Length	Over-fllow	Via Count	Routed Length
IBM01	102	15434	65588	0	17124	63332	255	15084	66058
IBM02	33	32529	178759	0	37937	168918	8	30668	174062
IBM03	0	25724	151299	0	31993	146412	0	22809	147524
IBM04	309	30836	173289	0	38464	167101	618	28611	172652
IBM05	0	51228	409747	0	77104	409739	0	50321	409778
IBM06	0	45692	282325	0	57036	277608	0	42847	280007
IBM07	53	60832	378876	0	78563	366180	0	56895	381694
IBM08	0	75291	415025	0	93905	404714	0	69321	413300
IBM09	0	68707	418615	0	86645	413053	0	64419	416554
IBM10	0	100546	593186	0	128141	578795	0	95316	591036
Average		+6.4%	+0.5%		+35.8%	-1.9%			

Table 3.2: Solution quality comparison of Sidewinder to BoxRouter 1.0 [20] and FGR 1.0 [93]. Note that on these benchmarks, unlike the ISPD 2007 benchmarks, the default mode of FGR 1.0 does not penalize bends and only minimizes wirelength without accounting for vias.

percentage of unrouted nets after ILP are trivial and a detail router is able to compensate (Fig. 2.3). In order to compare directly with BoxRouter 1.0 and FGR 1.0, we take the solutions generated by Sidewinder and route all remaining unrouted nets with a *single pass* of a maze router (no nets originally routed were ripped-up).

Table 3.2 compares these fully routed solutions to those of FGR 1.0 and BoxRouter 1.0 in terms of total overflow, via count and total routed wirelength. We first compare against FGR 1.0 [93], which won the ISPD 2007 Contest [51] in the 2D Category. While FGR 1.0 completes all the ISPD98 benchmarks without violation, its via counts are higher than

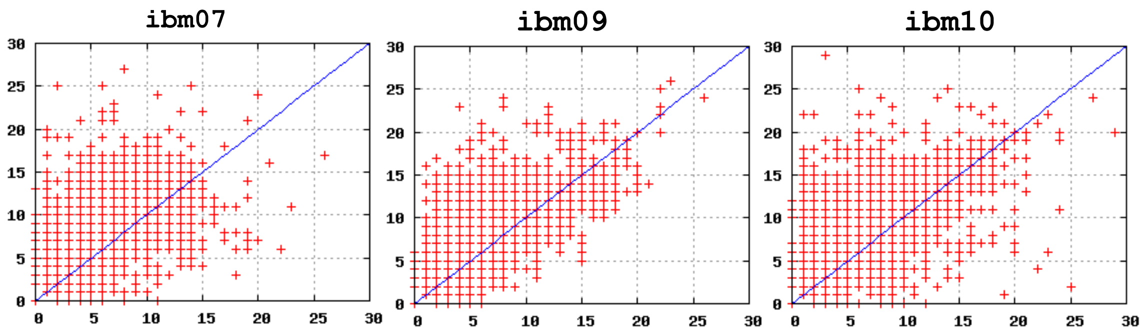


Figure 3.3: Via count comparison between Sidewinder and BoxRouter 1.0 for (a) IBM07, (b) IBM09, and (c) IBM10. The x- and y-axes state the number of vias for Sidewinder and BoxRouter 1.0, respectively. Each net is represented by a point whose coordinates are the number of vias it has in the results of these two routers. The blue line shows where Sidewinder and BoxRouter 1.0 use the same number of vias for a given net. Thus, if a point is above the blue line, Sidewinder uses fewer vias than BoxRouter 1.0 for the same net.

Sidewinder's by 35.8%. Note that since this set of benchmarks don't formally have vias, we refer to vias as when a net "bends". That is, a via is counted when a horizontal routing segment is followed by a vertical segment (or vice versa). FGR 1.0, in this case, did not penalize bends.

Compared against BoxRouter 1.0, we achieve 6.4% less vias and 0.5% shorter routed wirelength with moderate amounts of overflow. The via comparison is further depicted in Figure 3.3. The blue line represents where both routers use the same number of vias for that net. That is, a data point above the blue line means Sidewinder uses fewer vias and a data point below the blue line indicates Sidewinder uses more vias. Against BoxRouter 1.0, Sidewinder uses fewer vias on the vast majority of the nets. Using more sophisticated techniques such as iterations of rip-up and reroute, we could improve these violation counts. However, Sidewinder's solutions are sufficient to be used by a detail router.

3.4 Conclusions

In this chapter, we propose the first ILP router that can handle the entire global routing grid and produces routing solutions with very few vias. Our route selection algorithm is congestion-driven - during each iteration, the algorithm intelligently selects the two best (least congested) routes as candidates based on a dynamically updated congestion map. Our ILP formulation is scalable: for a net $n \in N$, we only consider two possibilities. Thus, given $|N|$ nets, we only need $2|N|$ variables. In addition to the traditional L and Z routing patterns, we introduce shapes with detouring, C shapes, to significantly improve routability. Our formulation guarantees that each new solution will be no worse than the current solution. Note that our incremental ILP approach is not limited to global ASIC routing – it is adaptable to detailed routing and FPGA routing.

Our ILP formulation can be easily extended to various aspects of detail routing. In particular, mutual exclusion constraints and logical implications can be expressed compactly. This type of framework allows designers to easily handle complex design rules. One important application is forbidden pitches - routes must comply with multiple distance bounds with respect to each other. For example, two routes can be $[2\lambda, 4\lambda]$ or $[8\lambda, 10\lambda]$ apart;[2] $(4\lambda, 8\lambda)$ is strictly forbidden. Typically, complying with these restrictions is incredibly difficult, as there is no efficient way of modeling these limitations. However, using ILP, these design constraints be easily expressed and easily integrated with other design rules and models (and solved optimally). Assignment of routing tracks in detailed routing can also be expressed efficiently using ILP formulations similar to ours. As demonstrated in Section 3.3, Sidewinder is adept at handling arbitrary pricing for vias.

[2] λ is a technology-dependent distance metric.

CHAPTER IV

Completing High-quality Global Routes

In this chapter, we expand on the capabilities of iterative global routers. Several iterative global routers have emerged after the ISPD 2007 and 2008 Global Routing Contests [51, 84], significantly improving the state of the art. Through empirical validation, the top-performing global routers are based on *negotiated-congestion*, where, instead of accounting for only the present state, global routers account for both the present and *the past*. Our research expands upon this foundation, and introduce several improvements that are applicable to any generic global routing framework, where (i) nets are first decomposed into two-pin subnets, (ii) all subnets are routed on a two-dimensional grid, and (iii) the subnets are reconstituted and projected on the three-dimensional grid. Our goal is to develop a high-quality router that, given a set of constraints, finds the optimal (or near-optimal) locations for each net. Using Lagrangian relaxation, we focus on routing segments, and assign *edge-centric* history costs, instead of *net-centric* history costs. This gives the router additional freedom when finding low-cost paths. Empirically, we produce high-quality routing solutions *without* tailoring them to specific benchmarks.

4.1 Introduction

High-quality routing solutions on recent large-scale benchmarks [51] from IBM were produced by FGR 1.1 [93]. At the ISPD 2008 Global Routing Contest [84], NTHU-Route 2.0 [14] and NTUgr [16] also posted high-quality results, along with improved runtimes. In addition, FastRoute 4.0 [121] claimed exceptionally low runtimes.

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 [84]. 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 produced by NTHU-Route 2.0 [14] with more violation-free solutions and comparable wirelength. BFG-R also produces better solutions than NTUgr [16] and FastRoute 4.0 [121] 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 chapter is structured as follows. Section 4.2 outlines BFG-R's global routing flow. Section 4.3 describes the new algorithms that are key to BFG-R's

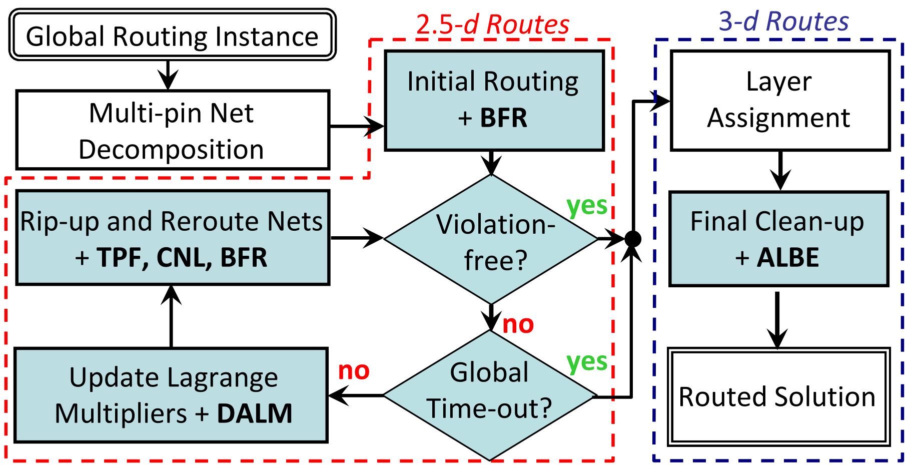

Global Routing Instance
Multi-pin Net Decomposition
2.5-*d* Routes
Initial Routing + **BFR**
Rip-up and Reroute Nets + **TPF, CNL, BFR**
Violation-free?
yes
no
Update Lagrange Multipliers + **DALM**
no
Global Time-out?
yes
3-*d* Routes
Layer Assignment
Final Clean-up + **ALBE**
Routed Solution

Figure 4.1: 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).

high performance. Section 4.4 discusses the data structure that allows BFG-R to maintain scalability for large problem instances and handle large netlists. Section 4.5 presents BFG-R's results on the ISPD08 Contest benchmarks [84] and ADAPTEC netlists re-placed with mPL6 [13]. Section 4.7 concludes our current work and discusses future work.

4.2 Global Routing Framework

In this section, we explain the general global routing framework of BFG-R, as shown in Figure 4.1. 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, e.g., 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 predetermined 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.

4.2.1 Multi-pin Net Decomposition

Competitive routers explicitly decompose (split) large nets into sets of two-pin subnets. There are two mainstream methods: (i) minimal spanning tree (MSTs), used by NTUgr [16] and FGR 1.1 [93], and (ii) Steiner minimal trees (SMTs), used by NTHU-Route 2.0 [14] and FastRoute 4.0 [121]. 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 4.4.1, 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.

4.2.2 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 [82, Section 3.4] by tuning weights in linear combinations. However, as stated in [21], *ad hoc* tradeoffs may lead to violent divergence of routing iterations. Therefore, several routers use *dampening* factors to ensure convergence [21].

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 [93]. 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 [71] 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 [93]:

$$c_e = b_e + h_e \cdot p_e \tag{4.1}$$

Lagrange multipliers are updated at the beginning of rip-up and reroute iteration k [93]:

$$h_e^k = \begin{cases} h_e^{k-1} + h_{step} & \text{if } e \text{ is over-capacity} \\ h_e^{k-1} & \text{otherwise} \end{cases} \tag{4.2}$$

Compared to previous work, we use a different penalty function p_e for local congestion (Section 4.3.3), and we do not use a constant h_{step} (Section 4.3.2). The stopping criterion for rip-up and reroute iterations gauges the amount of effort applied on hard-to-route instances. In our current implementation, BFG-R stops when a legal solution is found or running for 24 hours (according to the ISPD08 Global Routing Contest [84]).

4.2.3 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*.

4.2.4 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 [14] and Sidewinder [42], and monotonic maze routing, used by FastRoute 2.0 [88].

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, Z, and C 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 [88]. 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 (i) combines Dijkstra's shortest path algorithm with a non-trivial lower-bound function,[1] (ii) 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.

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

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.

4.2.5 Continuous Net Restructuring

Published competitive routers, e.g., NTHU-Route 2.0 [14], NTUgr [16], FastRoute 4.0 [121] and FGR 1.1 [93], all employ net restructuring during maze routing. To preserve topological flexibility, we restructure nets continually similar to FGR 1.1. To limit runtimes, we develop Cyclic Net Locking (CNL), described in Section 4.3.5 below.

4.2.6 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 constructed from the 2-d solutions.

The authors of [93] 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 [21], dynamic programming [73], and a greedy approach [93]. 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 (i) edges that have violations are already illegal and (ii) 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 our 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.

4.3 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.

4.3.1 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.

4.3.2 Dynamically Adjusting Lagrange Multipliers (DALM)

Updating history costs is critical to the success of the negotiated-congestion [80] and discrete Lagrange multiplier [93] routing frameworks. They must be precisely determined, as they are the dominant factors in determining both solution quality and runtime.

Previous work [80,93] increases Lagrange multipliers of congested edges by a constant h_{step} according to Equation 4.2. 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:

$$h_e^k = \begin{cases} h_e^{k-1} + h_{step} \times 1.25 & \text{if } e_{OF} \geq TEdge_{OF} \\ h_e^{k-1} + h_{step} & \text{else if } e_{OF} > 0 \\ h_e^{k-1} & \text{otherwise} \end{cases} \tag{4.3}$$

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:

$$h_e^k = h_e^k + (1 - cluRatio + \alpha)^{-1} \text{ if } e_{OF} \geq TClu_{OF} \tag{4.4}$$

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 use $\alpha = 0.75$ 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 $[h_{step}^{min}, h_{step}^{max}]$ over time. Our initial step is chosen to be $\frac{h_{step}^{max}+h_{step}^{min}}{2}$, and we choose a *delta* for Lagrange steps $\Delta_{step} = \frac{h_{step}^{max}-h_{step}^{min}}{200}$. We route in the framework of Section 4.2 and Figure 4.1, while Lagrange steps are modified between iterations as follows

$$h_{step}^{k+1} = \begin{cases} h_{step}^{k} + \Delta_{step} & \text{if } viol_k \geq viol_{k-1} \\ & \text{if } viol_k < viol_{k-1} \text{ and} \\ h_{step}^{k} - \Delta_{step} & WL_k > WL_{k-1} \\ & \text{if } viol_k < viol_{k-1} \text{ and} \\ h_{step}^{k} & WL_k \leq WL_{k-1} \end{cases} \tag{4.5}$$

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 then increase when nearing a legal solution. As reported in Table 4.2, this technique helps BFG-R find high-quality routes while reducing violation counts.

4.3.3 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. [36] amplified the congestion cost at a linear rate, capping the

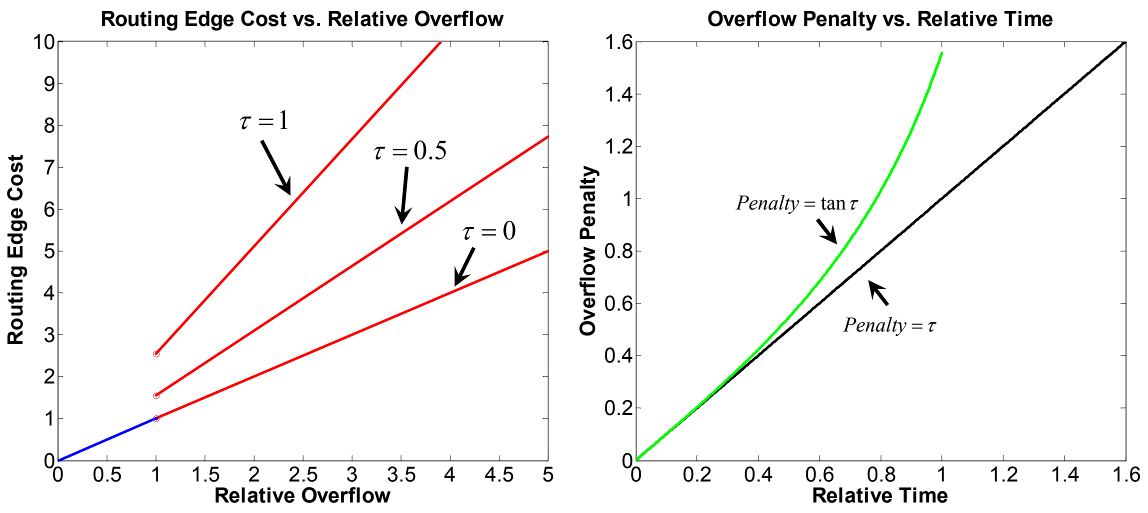


Figure 4.2: Trigonometric cost function used in BFG-R. The overflow penalty grows trigonometrically with the relative time τ (left). The cost function grows linearly with overflow (right).

growth at 1.2× after the routing edge was over- capacity by 20%; FGR 1.1 [93] and NTHU 2.0 [14] 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 $\tau = \frac{CurrentTime}{MaxTimeAllowed}$

$$p(e) = \begin{cases} \omega_e \times (1 + \tan(\tau)) & \text{if } \omega_e > 1 \\ \omega_e & \text{otherwise} \end{cases} \tag{4.6}$$

BFG-R's cost function grows linearly with overflow but trigonometrically with time, as shown in Figure 4.2. 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.

4.3.4 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 [14] or use benchmark-specific fixed costs [121].

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

$$p(VIA) = \lceil l/2 \rceil \times viaFactor \tag{4.7}$$

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

4.3.5 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 GCells so that long flat subnets do not have zero area.

$$BB\_Area(n) = (ur.x - ll.x + 1) \times (ur.y - ll.y + 1) \tag{4.8}$$

Here, ll and ur represent the lower left and upper right coordinates of n's bounding box.

From this, we found that (i) almost all of the nets' routes are within $2\times$ of their HPWL and (ii) 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:

$$AvgArea = \frac{1}{N}\sum_{n=1}^{N} BB\_Area(n) \tag{4.9}$$

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

$$Period(n) = \min\left(\left\lceil \frac{BB\_Area(n)}{AvgArea} \right\rceil, 20\right) \tag{4.10}$$

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

$$(i < 2) \text{ or } ((i + n) \mod Period(n) = 0) \tag{4.11}$$

This condition effectively staggers unlocking of large nets and also allows them to be unlocked with the proper period. We find that this method improves the framework of Section 4.2 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.

4.3.6 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

[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.

into Dijkstra's algorithm if its admissible function underestimates the true path-suffix cost. This effect is pronounced when (i) temporarily setting shared edge costs to zero when routing multi-pin subnets and (ii) 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 [93] 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.

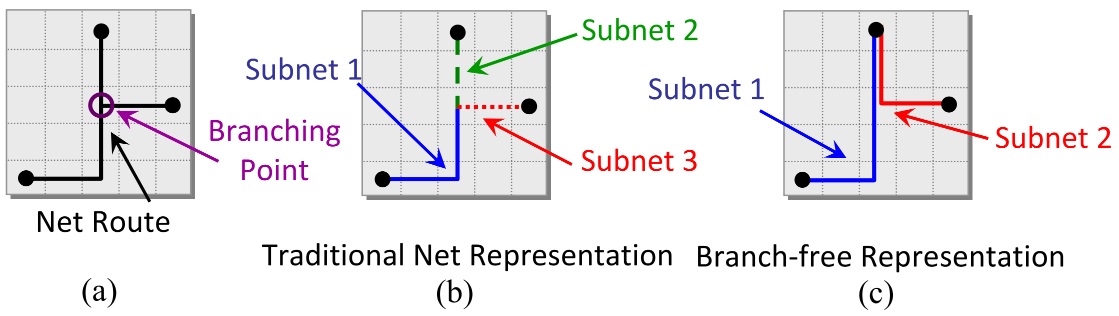


Figure 4.3: The branch-free representation (BFR) of routed nets. Subnets are treated separately but can share routing edges. Collectively they represent a Steiner tree.

4.4 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× less than NTHU (4× less on the largest benchmarks).

4.4.1 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 4.3, 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 [88] and NTHU-R 2.0 [14]. Other routers like MaizeRouter [81] 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 [93]. 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 (i) routing edge indices and (ii) 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 (i) the overlap in BFR between subnets is small, as long as the net is initially decomposed using an MST and (ii) coalescing subnets takes little time. Other routers, on the other hand, choose to use more memory to reduce runtime. For example, NTHU-R [14] uses large hash maps and pre-computes edge costs for constant-time lookup.

	NTHU-Route 2.0 [14]			NTUgr [16]			FastRoute 4.0 [121]			BFG-R		
Bench-mark	OF Total	Cost (e6)	Time (min)	OF Total	Cost (e6)	Time (min)	OF Total	Cost (e6)	Time (min)	OF Total	Cost (e6)	Time (min)
Solution Quality and Runtime for ROUTABLE Benchmarks												
A1	0	5.37	6.4	0	5.67	42.4	0	5.50	3.6	0	5.43	8.4
A2	0	5.24	2.8	0	5.47	7.4	0	5.28	1.2	0	5.23	3.7
A3	0	13.15	4.2	0	13.77	35.0	0	13.26	2.7	0	13.14	16.0
A4	0	12.18	15.1	0	12.41	14.7	0	12.15	1.1	0	12.16	5.2
A5	0	15.54	5.2	0	16.52	100.9	0	15.91	10.3	0	15.67	15.5
BB1	0	5.57	10.0	0	5.95	118.3	0	5.89	8.0	0	5.72	10.2
BB2	86	9.00	12.2	118	9.47	212.0	`Invalid Solution`			0	9.11	40.8
BB3	32	13.07	9.7	0	13.49	25.6	`MAZE RIPUP WRONG`			0	13.18	20.6
NB1	164	4.60	14.2	212	4.82	136.0	542	4.73	13.6	0	4.68	256.9
NB2	0	7.59	1.1	0	7.85	5.1	0	7.53	0.7	0	7.57	1.5
NB5	18	23.14	29.0	0	24.25	117.9	0	23.51	13.8	0	23.30	47.6
NB6	0	17.70	49.4	0	18.74	76.6	`MAZE RIPUP WRONG`			0	18.01	15.7
Fail.	4			2			4			0		
Impr.		0.99			1.04			1.01			1.00	
Solution Quality and Runtime for UNROUTABLE Benchmarks												
BB4	256	22.80	72.9	410	24.35	302.9	`Invalid Solution`			434	23.20	1416.6
NB3	`Time Out`			33636	11.00	163.6	38020	10.88	1344.1	33900	10.64	1420.9
NB4	222	12.89	31.2	284	13.89	223.3	212	13.16	27.7	218	13.08	1413.3
NB7	68	35.52	1284.6	906	36.91	1403.9	`Invalid Solution`			606	35.21	1421.1

Table 4.1: BFG-R compared with leading routers on the ISPD08 benchmarks [84] where A1 → ADAPTEC1, BB1 → BIGBLUE1, NB1 → NEWBLUE1, and so on. NTHU 2.0 is NTHU-Route 2.0 and FR 4.0 is FastRoute 4.0. Experimental setup is described in Section 4.5.1. `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 (i) some routers did not produce valid solutions on all benchmarks, (ii) some routers did not succeed on routable benchmarks, and (iii) benchmark solution quality varies significantly.

4.4.2 Representing a Dynamic Routing Grid

The main challenges when designing a data structure for a routing grid are (i) to keep the structure slim to improve cache locality (thereby reducing runtime) while fitting large instances into the 32 (64)-bit address space, and (ii) to provide constant-time access to the grid cells and routing edges. Our routing grid consists of an array of routing tiles connected by routing edges. Each routing tile contains six indices which represent the six routing edges (two each in x, y, and z directions) to which it can be connected. Tiles are stored such that the index of the tile in the array is calculated in constant time from the x, y, and z coordinates of the tile and vice versa. Thus, memory is saved by not requiring tiles to store their coordinates. In each routing edge, we store: (i) its type (`VIA`, `HORIZONTAL`, or `VERTICAL`), (ii) its metal layer, (iii) its Lagrange multiplier (described in Section 4.2.2), (iv) the routing resource capacity, (v) the current resource usage, and (vi) a list of the subnets that pass through it.

Note that routing edges do not store additional information such as edge costs. There are three reasons for this. First, the functions we employ for determining edge costs can be computed quickly and dynamically with the information currently stored on the edge. Thus, we save memory with minimal impact on runtime. Second, since we allow for the use of different cost functions, on-the-fly computation is more flexible. Third, in order

Benchmark	Best Tuned [14, 16, 121]			BFG-R (No Tuning)		
	OF Total	Cost (e6)	Router Name	OF Total	Cost (e6)	Time (m)
Solution Quality and Runtime for ROUTABLE Benchmarks						
ADAPTEC1	0	5.36	NTHU 2.0	0	5.43	8.4
ADAPTEC2	0	5.23	NTHU 2.0	0	5.23	3.7
ADAPTEC3	0	13.11	NTHU 2.0	0	13.14	16.0
ADAPTEC4	0	12.17	NTHU 2.0	0	12.16	5.2
ADAPTEC5	0	15.54	NTHU 2.0	0	15.67	15.5
BIGBLUE1	0	5.57	NTHU 2.0	0	5.72	10.2
BIGBLUE2	0	9.06	NTHU 2.0	0	9.11	40.8
BIGBLUE3	0	13.08	NTHU 2.0	0	13.18	20.6
NEWBLUE1	0	4.65	NTHU 2.0	0	4.68	256.9
NEWBLUE2	0	7.53	FR 4.0	0	7.57	1.5
NEWBLUE5	0	23.17	NTHU 2.0	0	23.30	47.6
NEWBLUE6	0	17.70	NTHU 2.0	0	18.01	15.7
Rout. Failures	0			0		
Improv. 0 OF		0.99			1.00	
Solution Quality and Runtime for UNROUTABLE Benchmarks						
BIGBLUE4	162	23.10	NTHU 2.0	434	23.20	1416.6
NEWBLUE3	31106	17.15	NTUgr	33900	10.64	1420.9
NEWBLUE4	138	13.04	NTHU 2.0	218	13.08	1413.3
NEWBLUE7	54	35.58	FR 4.0	606	35.21	1421.1

Table 4.2: BFG-R compared with the best-reported results on the ISPD08 benchmarks [84], where NTHU 2.0 is NTHU-Route 2.0 and FR 4.0 is FastRoute 4.0. Experimental setup is described in Section 4.5.1. Runtimes are not averaged because (i) some routers did not produce valid solutions on all benchmarks, (ii) some routers did not succeed on routable benchmarks, and (iii) benchmark solution quality varies significantly.

to support nets with different wire widths and layers with different routing pitches, costs cannot be computed *per edge* and stored statically.

4.4.3 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. Then, every routing edge e is removed from the list of used edges maintained by s.

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. Then, every routing edge e adds s to its list of subnets.

4.5 Empirical Evaluation

First, we describe our experimental setup and highlight relevant information about the benchmarks used; the full set of information is found in Table 4.3. Next, we compare our solution quality on those benchmarks against the top three performers from the ISPD 2008 Global Routing Contest [84].

Benchmark	Grid ($X \times Y$)	# Nets	# Layers	H. Cap	V. Cap	Routable?
A1 †	324 × 324	219794	6	70	70	yes
A2 †	424 × 424	260159	6	80	80	yes
A3 †	774 × 779	466295	6	62	62	yes
A4 †	774 × 779	515304	6	62	62	yes
A5 †	465 × 468	401060	6	110	110	yes
BB1	227 × 227	282974	6	110	110	yes
BB2	468 × 576	576816	6	52	52	yes
BB3	555 × 558	1122340	8	148	148	yes
BB4	403 × 405	2228903	8	204	240	no
NB1 †	399 × 399	331663	6	62	62	yes
NB2 †	557 × 463	463213	6	110	110	yes
NB3 †	973 × 1256	551667	6	80	80	no
NB4	455 × 458	636195	6	80	80	no
NB5	637 × 640	1257555	6	88	88	yes
NB6	463 × 464	1286452	6	132	132	yes
NB7	488 × 490	2635625	8	210	210	no

Table 4.3: General statistics on the ISPD08 benchmarks [84]. † indicates that it was part of the ISPD07 benchmark suite [51].

4.5.1 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 [14] 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[3], NTUgr's [16] C++ code with g++ 4.3.2, and FastRoute 4.0's [121] C code with gcc 4.3.2.

For an objective comparison, we ran each router, including BFG-R, in its default mode,

[3]NTHU-Route 2.0 is currently incompatible with g++ 4.3.2.

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. We imposed a time-out limit of 24 hours. This limit was respected in all cases, except for NTHU-Route 2.0 running `newblue3`.

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, NTHU-Route is invoked by a Perl script that uses a different set of parameters for each benchmark. To untune NTHU-Route 2.0, we use its default settings. NTUgr used the number of non-trivial nets to differentiate between benchmarks and ran tailored flows with pre-set thresholds. To untune NTUgr, we used a level that did not match any benchmark in the ISPD08 contest benchmark suite. FastRoute 4.0 used a set of specific parameters based on the benchmark name. To untune FastRoute 4.0, we renamed the input files.

4.5.2 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 [13], 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 4.4) are transitional values for which the benchmarks become routable – increasing the target density would lead to routability problems.

4.5.3 Comparison of Results

In our experiments, each router was configured with identical parameters for all benchmarks. Table 4.2 compares BFG-R's performance on ISPD 2008 Contest benchmarks.

	NTHU-Route 2.0 [14]			NTUgr [16]			FastRoute 4.0 [121]			BFG-R		
Bench-mark	OF Total	Cost (e6)	Time (min)	OF Total	Cost (e6)	Time (min)	OF Total	Cost (e6)	Time (min)	OF Total	Cost (e6)	Time (min)
A1, 70%	0	4.62	7.2	0	4.83	73.2	184	5.01	26.4	0	4.68	9.8
A2, 60%	0	5.29	0.9	0	5.48	3.7	0	5.31	0.6	0	5.28	2.2
A3, 80%	38	12.16	19.4	28	12.88	470.0	616	12.74	183.1	0	12.15	27.2
A4, 80%	0	10.50	2.3	0	10.75	9.1	10	10.61	4.8	0	10.49	3.2
A5, 70%	4	13.91	25.2	0	14.44	347.8	628	14.49	50.6	0	13.98	32.6
Fail.	2			1			4			0		
Impr.		1.00			1.03			1.01			1.00	

Table 4.4: BFG-R compared with leading routers on the re-placed ADAPTEC benchmark suite. Each benchmark's netlist was placed using mPL6 [13] with its corresponding target density. These benchmarks were not used during the development of the routers we evaluate.

Similarly, Table 4.4 compares BFG-R's performance on the re-placed ADAPTEC suite. The 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 [14], NTUgr [16], and Fast-Route 4.0 [121]. 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, NEWBLUE4, and NEWBLUE7.[4] Every design in the second set was shown to be routable.

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.

[4] 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.

Wirelength. As illustrated in Table 4.2, 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.

4.6 Scalability Study

To test the scalability of BFG-R and predict its effectiveness for trillion-gate systems, we modify the DAC 2012 Routability-driven Contest benchmarks [109] to include two and three times the number of nets. Table 4.6 reports the runtime of BFG-R under the official DAC and ICCAD 2012 Routability-driven Placement Contest [109,110] evaluation settings. Experiments were performed on an 3.4GHz Intel Xeon workstation. The average relative required runtime scales linearly with the number of nets, where we achieve a competitive routing speed of 5K nets per second per thread [77].[5]

4.7 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 [14], NTUgr [16], and FastRoute

[5]Based on median performance across all benchmarks.

				BFG-R Runtime (s)		
Benchmark	# Nets 1×	# Nets 2×	# Nets 3×	1×	2×	3×
SUPERBLUE2	990K	1.98M	2.97M	765	1262	1916
SUPERBLUE3	898K	1.80M	2.69M	373	702	1062
SUPERBLUE6	1.00M	2.01M	3.02M	237	464	778
SUPERBLUE7	1.34M	2.68M	4.02M	162	303	443
SUPERBLUE9	834K	1.67M	2.50M	126	212	300
SUPERBLUE11	936K	1.87M	2.81M	130	239	353
SUPERBLUE12	1.29M	2.59M	3.88M	253	510	715
SUPERBLUE14	620K	1.24M	1.86M	106	218	311
SUPERBLUE16	698K	1.40M	2.09M	190	331	487
SUPERBLUE19	512K	1.02M	1.54M	38	70	97
Average (×)				1.00×	1.85×	2.72×

Table 4.5: Runtimes of BFG-R [43] on DAC 2012 benchmarks [109] with the original netlist (1×), two times the original size (2×), and and three times the original size (3×). Experiments were performed with an 3.4GHz Intel Xeon CPU.

4.0 [121]. 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 rout-ed 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 predetermined 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 tradeoffs 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 wire widths. *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.

CHAPTER V

A SimPLR Method for Routability-driven Placement

Building upon our experience with standalone global routing, we now directly address the challenges of routability-driven placement within a simultaneous place-and-route framework. Since global routing is now used to guide global placement, the emphasis on solution quality is lessened; in turn, the priority on runtime is increased, as the global router is invoked multiple times, but fidelity must be maintained. To do this, we use existing state-of-the-art tools, and determine the right balance between quality and runtime to create a full place-and-route framework. We develop *lookahead routing* to give the placer advance, firsthand knowledge of trouble spots, not distorted by crude congestion models. We extend placement to (i) spread cells in congested areas, and (ii) move cells together in less-congested areas to ensure short, routable interconnects in moderate runtime.

5.1 Introduction

Highly-optimized placements may lead to irreparable routing congestion due to inadequate models of modern interconnect stacks and the impact of partial routing obstacles. Additional challenges in routability-driven placement include scalability to large netlists and limiting the complexity of software integration.

First, we develop *lookahead routing*, which invokes the global router to quickly estimate routability. Since the produced routes can be used as a routing solution, our method can accurately and quickly report both congestion and routed wirelength. **Second**, to produce competitive placements in terms of both routed wirelength and HPWL, we integrate our lookahead routing into a flat, quadratic global placer, and enhance placement iterations by gently coercing cell locations and relieve congestion while preserving interconnect length. In detailed placement, we do not change the objective functions as in [126], but *prohibit moves* that aggravate routability. In global placement, we temporarily inflate cells in highly-congested regions to reserve whitespace during global placement. Traditionally, this has been accomplished either by cell bloating [9, 39, 95] during/after global placement, or by whitespace allocation [75, 94, 123] after placement. We observe that wirelength-driven global placers typically limit area utilization by a given amount through the entire layout based on target density. Therefore, in addition to *cell bloating*, we *dynamically adjust the target density* based on total routed wirelength.[1] This technique preserves overall solution quality and allows the placer to move cells in uncongested regions closer. **Third**, we develop a simultaneous place-and-route framework for global placement as well as a routability-driven detailed placement algorithm.

Our proposed methodology offers several advantages. Since we use a global router to estimate congestion, the routes for all nets are known. Moreover, by enabling the global placer to directly redistribute whitespace in response to routing congestion, we establish a more precise feedback loop (compared to add-on techniques proposed previously).

[1]Partitioning-based placers can adjust target density on a per region basis [9, 94]. In force-directed placers, this feature is more difficult to implement and seems unnecessary.

Our key contributions include:

- A method to control routability within the global placer while preserving solution quality by dynamically adjusting the global target density
- An effective cell-bloating technique by dynamically adjusting cell width based on design's perceived difficulty
- A simultaneous place-and-route framework
- A congestion-aware detailed placement algorithm that moves cells only when this does *not* hurt routability
- Empirical results on the ISPD 2011 contest benchmarks [108] that outperform every competitor on every benchmark with an average 2.04× improvement (Table 5.2) and a greater improvement 8.43× (Table 5.3) with a stronger global router.

The remainder of this chapter is structured as follows. In Section 5.2, we review the baseline algorithms that we use. In Sections 5.3 and 5.4, we introduce several new techniques to improve the routability of placements. In Section 5.5, we empirically validate proposed algorithms. Section 5.6 concludes our work and discusses its impact.

5.2 SimPLR

Our implementation uses the SimPL [66] global placer to quickly produce a tentative solution. We then apply lookahead routing by calling a modified version of the BFG-R [43] global router to estimate routing congestion and wirelength. We use this information in global placement by means of *cell bloating* and *dynamically adjust the target density*. After several iterations of global placement, performed using a modified SimPL placer, lookahead routing is invoked again. Such combined place-and-route iterations continue until convergence. Then, our modified version of the FastPlace-DP [89] detailed placer applies *congestion-aware detailed placement* to recover whitespace and improve routed wirelength while maintaining routability.

We now briefly review the baseline place-and-route algorithms. For further background on placement see [59, Chapter 4] and for global routing see [59, Chapter 5].

SimPL [66] is a flat, force-directed global placer. It maintains a lower-bound and an upper-bound placement; the final solution is generated when the two placements converge.[2] The upper-bound placement is generated by applying *lookahead legalization (LAL)*, which is based on top-down geometric partitioning and non-linear scaling. Using this information, the lower-bound placement is generated by minimizing the quadratic objective

$$\Phi_q(\vec{x},\vec{y}) = \sum_{i,j} w_{i,j} \left((x_i - x_j)^2 + (y_i - y_j)^2\right) \tag{5.1}$$

using the Conjugate Gradient method [96]. Here, $\vec{x}$ and $\vec{y}$ are coordinate vectors of the cells' (x,y) locations, and $w_{i,j}$ represents the connectivity between cells i and j. Two of top three teams in the ISPD 2011 contest, including ours, relied on the SimPL algorithm.

FastPlace-DP [89] is a wirelength-driven detailed placer based on greedy algorithms. It uses (i) cell clustering, (ii) global cell swapping, (iii) vertical cell swapping, and (iv) local reordering to improve wirelength. To determine which cells should be swapped, FastPlace-DP estimates the reduction in wirelength from swapping cells i and j by

$$gain(i,j) = \sum_{n\in N_i} (W_n - W'_n) - \sum_{n\in N_j} (W_n - W'_n) \tag{5.2}$$

where N_i and N_j are the nets connected to cells i and j, and W and W' are the wirelength measurements before and after the swap.

[2]The wirelength gap between the upper-bound and lower-bound solutions is useful to formulate convergence criteria.

BFG-R [43] is a global router based on Lagrangian relaxation. It decomposes multi-pin nets into two-pin subnets using MSTs and then iteratively routes all subnets until no violations are present. BFG-R prices each (sub)net by summing up the cost of used edges

$$cost(e) = base_e + \lambda(e) \times C(e) \times \rho(e) \tag{5.3}$$

where $base_e$ is the base edge cost, $\lambda(e)$ is the history cost, $C(e)$ is current congestion, and $\rho(e)$ is the runtime penalty factor.

5.3 Simultaneous Place-and-Route

In this section, we introduce a methodology for simultaneous place-and-route and discuss its components (Figure 5.1). Given a placement instance, the baseline global placer quickly produces a tentative solution. Then, we apply *lookahead routing (LAR)* by calling our global router to estimate routing congestion and wirelength. We use this information during global placement by means of *cell bloating* and *dynamically adjusting the target density*. After several iterations of global placement, where the placer "heals" the placement for wirelength, lookahead routing is invoked again, and such iterations continue until overflow stops improving. *Congestion-aware detailed placement* is performed to recover whitespace and improve routed wirelength while maintaining routability.

We achieve an initial placement solution once the wirelength gap between the upper-bound and lower-bound solutions is within 25% of the 10^{th} iteration's total wirelength (Section 5.2). After cell bloating, we run four iterations of lookahead legalization. Our disjunctive convergence criterion checks for three conditions: (i) the overflow has improved less than 3% after two consecutive rounds of LAR, (ii) the total overflow is less than 1% of the total edge capacity, or (iii) the global placement time-out of 60 iterations.

5.3.1 Lookahead Routing (LAR)

To improve routability while preserving wirelength, global placement invokes lookahead routing. Unlike previous approaches, where only congestion information is reported, LAR estimates *both* interconnect length and routing congestion. Our router implementation accounts for (i) different wire widths and spacings, (ii) routing blockages, (iii) pins on different metal layers, and (iv) vias.

Wire widths and spacings at each metal layer are modeled separately. The resources consumed by a net n are then estimated by

$$Usage(n) = \sum_{e \in n} minSpacing(l_e) + minWidth(l_e) \tag{5.4}$$

where e is each edge in n, l_e is the metal layer that e is on, and $minSpacing(l_e)$ and $minWidth(l_e)$ are the respective minimum spacing and width required for l_e.

However, congestion estimates produced by this model can be misleading. For example, suppose two edges e_1 and e_2 on different metal layers are overflown, where e_1 is on Metal1, having $minSpacing(Metal1) + minWidth(Metal1) = 2$, and e_2 is on Metal4, having $minSpacing(Metal4) + minWidth(Metal4) = 8$. Suppose e_1 has two violating nets, yielding an overflow of 4, and e_2 has one violating net, yielding an overflow of 8. These actual overflows are now misleading, as e_1 is considered more congested, but its overflow is lower than that of e_2. Therefore, to accurately report congestion, we *normalize* the capacity for every edge e on metal layer l by

$$nCap(e_l) = \frac{Cap(e_l)}{minSpacing(l_e) + minWidth(l_e)} \tag{5.5}$$

where $Cap(e_l)$ is the original capacity of edge e on layer l. Note that when normalizing capacity, we also normalized, where each segment is defined as one routing segment, regardless of the layer.

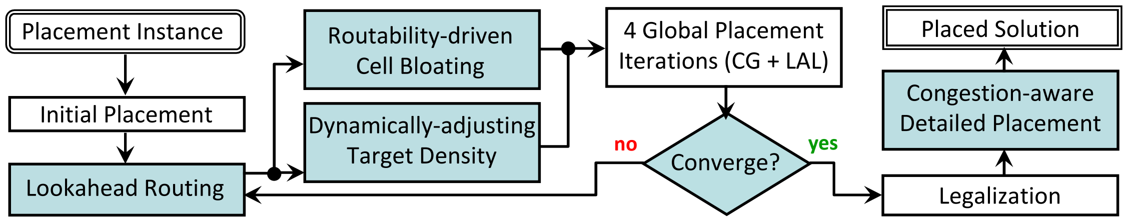

Placement Instance
Initial Placement
Lookahead Routing
Routability-driven Cell Bloating
Dynamically-adjusting Target Density
4 Global Placement Iterations (CG + LAL)
Converge?
no
yes
Legalization
Congestion-aware Detailed Placement
Placed Solution

Figure 5.1: Our simultaneous place-and-route (SimPLR) flow. The baseline components are shown in transparent boxes. Added routability-driven components have light-blue fill.

Routing blockages are specified as physical locations in the layout area. Therefore, the routing resources blocked at each edge are proportional to the length of the blockage. However, if two obstacles overlap, the overlap is only counted once. To properly calculate capacity, we first take the union of all routing-obstacle shapes on each edge, and then consider each non-blocked region separately. For each non-blocked region r, the amount of usable capacity is

$$v(r_e) = \frac{dim(r_e)}{dim(e)} \tag{5.6}$$

where $dim(r)$ is the length of the non-blocked region on edge e, and $dim(e)$ is the length of e (i.e., height if e is a vertical edge, and width if e is a horizontal edge). Then, each edge's normalized capacity is

$$nCap(e_l) = \sum_{r_e \in R_e} \frac{v(r_e) \times Cap(e_l)}{minSpacing(l_e) + minWidth(l_e)} \tag{5.7}$$

The calculation of normalized capacity in the presence of routing obstacles is demonstrated in Fig. 5.2. In the example, let the length of every edge be 50, let the lower left coordinate be (0,0). Let the original edge capacity be 40, and let the minimum spacing plus the minimum width of this layer be 4. Since the vertical edge (50,0)-(50,50) has coordinates (50,40)-(50,50) blocked off, it only has $\frac{50-10}{50} = 80\%$ usable capacity. Since there is only one non-blocked region, the normalized capacity is $\frac{(80\% \times 40)}{4} = 8$. Similarly, the

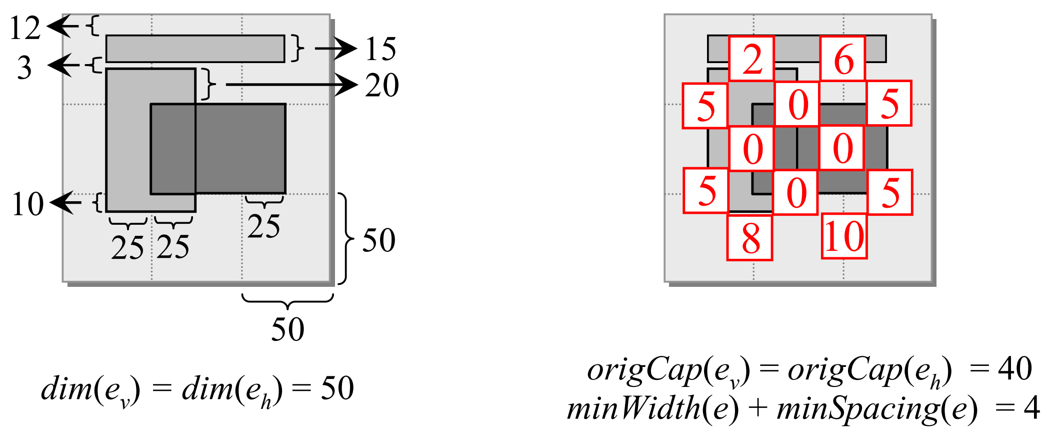

12
3
10
15
20
25 25
25
50
50
dim(e_v) = dim(e_h) = 50
2
6
5
0
5
0
0
5
0
5
8
10
origCap(e_v) = origCap(e_h) = 40
minWidth(e) + minSpacing(e) = 4

Figure 5.2: Accounting for routing blockages, where $dim(e) = 50$ for each edge, two of three routing blockages overlap. On the left, the lengths of each routing blockage and non-blocked region are shown. On the right, the normalized capacities are calculated for each edge. Here, the original capacity of each edge is 40, and each net on this layer uses 4 tracks. With no blockages, an edge has a normalized capacity of 10.

horizontal edge (50,50)-(100,50) has no usable capacity, as it is entirely blocked off, so its normalized capacity is 0.

Elevated pins. The ISPD 2011 contest benchmarks, derived from industrial ASICs and SoCs designs, include contact pins on multiple metal layers. This poses a challenge for traditional global routing techniques, where routing is first performed on a 2-d grid and then projected onto a 3-d grid. Therefore, we pursue a different strategy. We decompose all multi-pin nets into two-pin subnets, and perform 3-d maze routing.

5.3.2 Congestion-based Cell Bloating

After lookahead routing, we inflate all cells located in congested regions. The congestion at GCell g, located at (x,y), is

$$C(g(x,y)) = \frac{Usage(g(x,y))}{Cap(g(x,y))} \tag{5.8}$$

where $nUsage$ and $nCap$ are respectively the normalized usage and capacity at $g(x,y)$.

The usage at each GCell is defined as

$$\begin{aligned} Usage(g(x,y)) &= \max(nUsage(e(x\pm 1,y)), nCap(e(x\pm 1,y)))+ \\ &\quad \max(nUsage(e(x,y\pm 1)), nCap(e(x,y\pm 1))) \end{aligned} \tag{5.9}$$

and the capacity at each GCell is defined as

$$Cap(g(x,y)) = nCap\,(e(x\pm 1,y)) + nCap\,(e(x,y\pm 1)) \tag{5.10}$$

where $nUsage$ is the normalized usage for edge e, and $nCap$ is the normalized capacity for e. Therefore, if $C(g(x,y)) > 1$, then $g(x,y)$ is considered congested. If at least one of the neighboring edges is congested, then the GCell is considered congested. For every cell in each congested GCell, we apply cell bloating by setting the cell's new width to

$$\max(width(cell)+1, 1+\theta\cdot\Lambda(cell)\cdot C(g(x,y))\cdot deg(cell)) \tag{5.11}$$

where $width$ is the current width of $cell$, θ is an adaptive function (described below), Λ is the number of times the $cell$ has been in a congested GCell, and deg denotes the cell degree (the number of cell pins connected to wires).

Our cell bloating approach is inspired by CRISP [95], but differs in three major ways. **First**, we apply cell bloating *during global placement*, while CRISP bloats cells *after placement*. We can therefore perform large-scale changes and, in our experience, our placer better adjusts to bloated cells, resulting in a smaller wirelength penalty. **Second**, we use *GCell-centric congestion* estimation, while CRISP uses *edge-centric congestion* estimation with a pin-density map. Our style of congestion estimation improves integration with a global router. Pin density has been a popular estimation technique for designs with relatively few metal layers. However, with modern 9+ layer interconnect stacks, it primarily affects the success of *detailed routing*, which is orthogonal to our work.[3] **Third**, while

[3]CRISP could be applied after our techniques, but the improvements will not be detectable by our experimental configuration that uses only a global router.

CRISP relies on a constant θ, our θ is a routing-solution-dependent function (described below), and based on the design's estimated difficulty and its routability. The intuition is that if a design is difficult to place or route, cells in congested regions need additional whitespace. Therefore, cells in those regions should be more inflated. We define $\theta(G)$ as

$$\theta(G) = \max\left(0, \alpha \cdot \eta(G) \cdot \xi(G) + \beta\right) \tag{5.12}$$

where G is the set of all GCells, α and β are constants determined from linear regressions, $\eta(G)$ indicates how *hard* a design is (e.g., how much available routing capacity there is), which is relatively insensitive to the routed solution, and $\xi(G)$ indicates the routability of the design, and is based on lookahead routing. We define $\eta(G)$ as

$$\eta(G) = \sum_{g \in G} \frac{Usage(g)}{Cap(g)} \tag{5.13}$$

and $\xi(G)$ as

$$\xi(G) = \frac{TOF(G)}{TCap(G)} \tag{5.14}$$

where $OF(G)$ and $Cap(G)$ are the respective total overflow and total capacity of all GCells in G. In our implementation, we empirically determined the values $\alpha = 0.017$ and $\beta = -0.01$ based on numerical regression (but not benchmark-specific tuning).

5.3.3 Dynamic Adjustment of Target Density

Target density (utilization) is one of the most critical parameters to trade off routability for wirelength in the final placement. However, finding the best target density for routability-driven placements remains an open problem. Unnecessarily high target density leads to better HPWL, but may also cause routing failures [1]. Lower target density, on the other hand, may increase the overall routed wirelength, which would lead to longer detours and consume more routing resources. By using a variable target density, we are trading off wirelength for routing demand in congested regions. As demonstrated in Figure

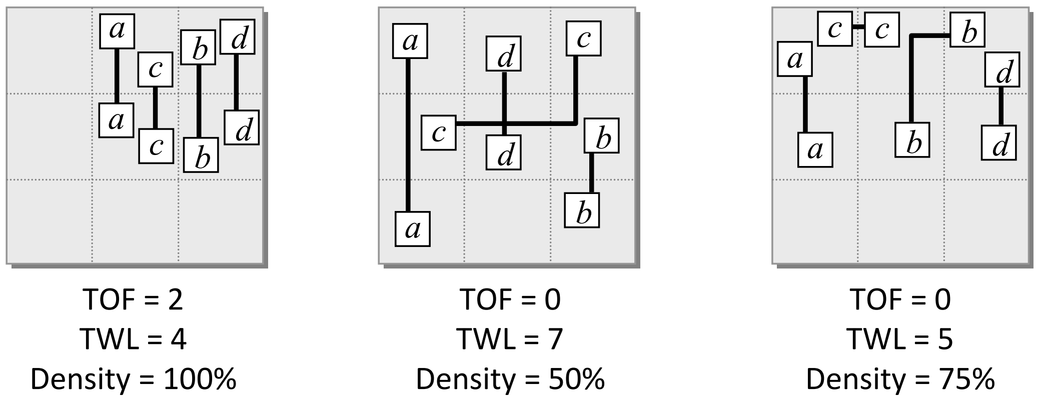


Figure 5.3: The impact of placement density on routability, with bin capacity 2 and edge capacity 1. The dense, low-wirelength placement (left) is unroutable. The sparse, high-wirelength placement (center) is routable. The placement (right) is also routable, with low wirelength and density.

5.3, a placer using a lower target density typically produces a placement that is more likely to be routable, but has higher total wirelength. Conversely, a placer using a higher target density typically produces a placement that is less likely to be routable, but has lower total wirelength. We set the initial target density as

$$\gamma_{init} = D_{ut} + \min(\max(\gamma_0 - D_{ut}, 0\%), \omega_D) \tag{5.15}$$

where D_{ut} is the design utility (given), γ_0 is a prediction of a *good* target density, and ω_D is the target density lower bound. If D_{ut} is too low (e.g., less than 35%), then the target density should be higher to encourage cell clustering. Conversely, cells should be spread apart if D_{ut} is too high. Empirically, we observed that setting $\gamma_0 = 50\%$ when $\omega_D = 15\%$ provides a reasonable tradeoff between routability and routed length.

After lookahead routing and cell bloating, the target density is updated as

$$\gamma = \min(\frac{area(C_m)}{area(D) - area(C_f)} + \phi, 95\%) \tag{5.16}$$

where C_m is the set of movable cells, C_f is the set of fixed cells, D is the design, $area$ returns the total area of input (bloated cells included), and ϕ is a constant that increases every time LAR reports an increase in routed wirelength. In our implementation, ϕ is initially $\gamma_{init} - D_{ut}$, and increases by 1% when wirelength increases.

5.4 Congestion-aware Detailed Placement

Traditional wirelength-driven detailed placement may pack cells in regions that are difficult to route. In the context of the FastPlace-DP Algorithm [89], we modify both global cell swapping and vertical cell swapping to be *congestion-aware* (Algorithm 6) in two ways: (i) we only allow cell move that do not harm routability, (ii) we encourage cells to move out of congested regions.

Algorithm 6 Congestion-aware Detailed Placement.

```
Input:   Set of all movable cells $C_m$, set of all congested cells $G_c$,
         and the congestion of every $i$ cell position $C(i)$
Output:  n.a.

1:  for all cells $c_i \in C_m$ do
2:      $R_i$ = optimal region of $c_i$;
3:      $b_{swap}$ = benefit of swapping $c_i$ with $c_j \in R_i$;
4:      $b_{move}$ = benefit of moving to an empty space $s \in R_i$;
5:      if $c_i \notin G_c$ AND $c_j \notin G_c$ then
6:          if $b_{swap} \geq b_{move}$ then
7:              PERFORM_SWAP($c_i$, $c_j$, ($b_{swap} > 0$));
8:          else
9:              PERFORM_SWAP($c_i$, $s$, ($b_{move} > 0$));
10:         end if
11:     else if $c_i \in G_c$ AND $c_j \notin G_c$ then
12:         PERFORM_SWAP($c_i$, $s$, TRUE);
13:     else if $c_i \notin G_c$ AND $c_j \in G_c$ then
14:         PERFORM_SWAP($c_i$, $c_j$, (DEGREE($c_i$) < DEGREE($c_j$)));
15:     else
16:         if $C(c_i) > C(c_j)$ then
17:             PERFORM_SWAP($c_i$, $c_j$, (DEGREE($c_i$) > DEGREE($c_j$)));
18:         else
19:             PERFORM_SWAP($c_i$, $c_j$, (DEGREE($c_i$) < DEGREE($c_j$)));
20:         end if
21:     end if
22: end for
```

The subroutine PERFORM_SWAP(c_i, c_j, $pred$) swaps two cells c_i and c_j if $pred$ is true.[4] For each movable cell c_i, we consider its best swap with c_j or move with empty space s (lines 2-4). If both actions result in positive gain, and both are in non-congested regions, then we revert to wirelength-driven decisions (lines 5-10). If c_i is in a congested region and c_j is not, then we can improve routability by moving it to s (lines 11-12). If c_i is not in a congested region and c_j is, and c_i has fewer pins than c_j, we can potentially improve routability in subsequent moves if we decrease the number of routes that go through the congested region (lines 13-14). Similarly, if both cells are in congested regions, then we only swap them if $deg(c_j) < deg(c_i)$. This ensures that the detailed placer does not harm routability (lines 15-20).

5.5 Empirical Validation

Our C++ implementation was compiled with g++ 4.4.3, and validated on a 3.00 GHz Intel Core 2 CPU X9650 Linux workstation. We modified and integrated the (i) SimPL global placer [66], (ii) BFG-R global router [43], and (iii) FastPlace-DP detailed placer [89]. Significant changes were made to all three tools, and new algorithms were added, as described in Sections 5.3 and 5.4.

The evaluation of placement solutions was performed by coalesCgrip [12], which was mandated by the ISPD 2011 Routability-driven Placement Contest [108]. coalesCgrip was compiled with gcc 4.4.1, as specified by the contest organizers. Its runtime limit was set to 300 seconds for initial routing and 900 seconds for rip-up and reroute (RRR), which makes results machine-dependent. Therefore, we downloaded all placements produced by the top contestants, and reevaluated them on our workstation.

Our implementation of SimPLR uses BFG-R for LAR instead of coalesCgrip, which

[4]A single cell can "swap" with an empty location.

BENCHMARK	#cells	FastPlace-DP (after SimPLR)				Ca-DP (after SimPLR)			
(source: IBM Research)		HPWL	RtWL	OF	Runtime	HPWL	ΔRtWL	ΔOF	Runtime
SUPERBLUE1	847K	**277.03**	**14.45**	**0**	**5.37**	279.01	0.376	**0**	9.83
SUPERBLUE2	1.01M	**657.03**	29.09	782348	**19.22**	660.09	**-0.195**	**-42298**	32.06
SUPERBLUE4	600K	231.78	10.71	22192	**2.96**	**231.44**	**-0.336**	**-3748**	4.62
SUPERBLUE5	772K	**354.23**	17.02	139012	**5.58**	355.05	**-0.386**	**-17118**	9.68
SUPERBLUE10	1.13M	**586.62**	**26.48**	**556678**	**7.99**	592.18	0.113	11102	18.26
SUPERBLUE12	1.29M	**376.59**	22.7	293516	**7.71**	377.27	**-0.119**	**-112166**	13.33
SUPERBLUE15	1.12M	**337.04**	**17.04**	56866	**6.60**	337.96	0.128	**-7580**	8.43
SUPERBLUE18	483K	**165.09**	10.64	23708	**2.92**	165.75	**-0.125**	**-2688**	4.44
Average		**1.00**×	**1.01**×	**1.18**×	**0.60**×	**1.00**×	**1.00**×	**1.00**×	**1.00**×
Geometric mean		**1.00**×	**1.01**×	**1.17**×	**0.60**×	**1.00**×	**1.00**×	**1.00**×	**1.00**×

Table 5.1: The impact of congestion-aware detailed placement on HPWL(×10e6), routed wirelength (×10e6), and overflow (OF) on ISPD 2011 benchmarks [108]. Runtimes are given in minutes. Routing was performed by *coalesCgrip* [12] with a 15-min time-out.

was not available in source code. Empirically, our router accurately predicts the regions of congestion reported by coalesCgrip while allowing us to implement our proposed interface that minimizes runtime overhead. Since our strong results are achieved *without* running coalesCgrip during global placement, SimPLR does not seem to require the knowledge of a specific downstream router. Though using different routers in one flow may not be ideal, this is not uncommon in multi-vendor industry flows, and our results indicate that such configurations can be successful.

Progress of global placement is illustrated in Figure 5.4 for the SimPL (with target density 50%) and SimPLR algorithms. Before the first invocation of lookahead routing (LAR) in SimPLR, the two progress identically since the initial target density (γ_{init}) of SimPLR is computed by Equation 5.15 to be 50%. The first invocation of LAR with subsequent cell bloating does not significantly impact wirelength, due to $\Lambda = 0$ in Equation 5.11. Lookahead legalization produces higher HPWL after the second LAR, but the impact on quadratic placement is small, and the disruption in roughly legalized placement is quickly healed. SimPLR invokes LAR three to six times per benchmark, taking roughly 27% – 58% of the total runtime, averaging at 47.88%, and currently runs $2\times$ slower than SimPL. Yet, SimPLR was among the two fastest placers at the ISPD 2011 contest.

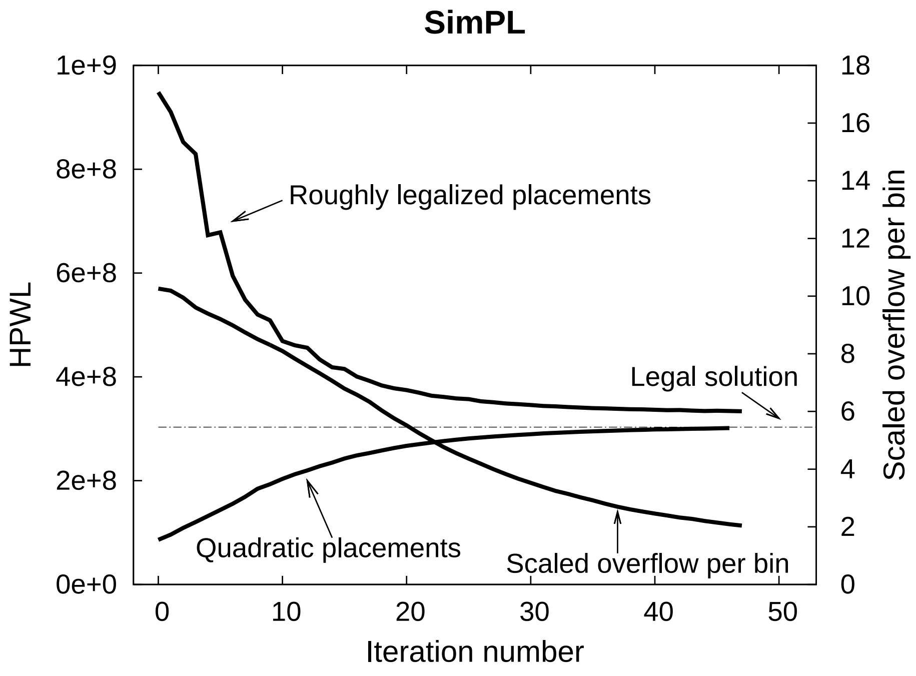
Congestion-aware detailed placement (Ca-DP) is evaluated in Table 5.1. We report the (i) recovered HPWL, (ii) recovered routed length, and (iii) total overflow improvement using Ca-DP, versus FastPlace-DP [89]. Ca-DP barely changes HPWL and routed wirelength, but improves overflow by $1.18\times$.

Routability is reported in Table 5.2 and Figure 5.5: SimPLR consistently reduces total overflow across all benchmarks and makes SUPERBLUE1 fully routable. On the remaining benchmarks, compared to baseline wirelength-driven placer SimPL, we improve total overflow by 3.81× on average. Compared to the top results from the ISPD 2011 Contest, we produce the smallest overflow on all benchmarks, for an average 2.04× reduction. These results are further improved in Table 5.3, and discussed below.

Extended runtime. The ISPD 2011 experimental protocol evaluated placements with only very short routing runs of coalesCgrip. To illustrate the full potential of SimPLR, we performed additional experiments, where coalesCgrip was given 12 and 24 hours. The results reported in Table 5.3 were obtained on a 2.53 GHz Intel Xeon CPU E5540 Linux workstation. While none of the IBM-released benchmarks could be completed without overflows at the ISPD 2011 contest, we have now completed half of them. Our results show that the advantage of SimPLR solutions grows significantly when the evaluating

BENCHMARK	#cells	SimPL with FastPlace-DP			Best in Contest		SimPL**R** with Ca-DP		
(source: IBM Research)		RtWL	OF	Runtime	RtWL	OF	RtWL	OF	Runtime
SUPERBLUE1	847K	**14.32**	1354	**23.89**	14.70	108	14.48	**0**	51.69
SUPERBLUE2	1.01M	**27.10**	1191806	**37.87**	30.77	797898	29.20	**740050**	108.30
SUPERBLUE4	500K	**10.52**	45430	**7.54**	10.86	85538	10.68	**18444**	24.79
SUPERBLUE5	772K	**16.90**	272934	**23.53**	17.29	126186	16.98	**121894**	51.84
SUPERBLUE10	1.13M	**26.18**	**463858**	**33.68**	25.16	616742	26.69	567780	73.34
SUPERBLUE12	1.29M	**19.35**	1992246	**35.18**	22.89	415428	22.58	**181350**	43.32
SUPERBLUE15	1.12M	17.09	62274	**24.21**	17.91	125936	**17.07**	**49286**	43.33
SUPERBLUE18	483K	10.64	153556	**14.36**	**9.84**	31440	10.63	**21020**	21.38
Average		**0.96**×	**3.81**×	**0.52**×	**1.01**×	**2.04**×	**1.0**×	**1.0**×	**1.0**×
Geometric mean		**0.96**×	**2.63**×	**0.49**×	**1.01**×	**1.76**×	**1.0**×	**1.0**×	**1.0**×

Table 5.2: Routed wirelength (RtWL, ×10e6), routing overflow (OF), and runtime (in minutes) on ISPD 2011 benchmarks. The placements were evaluated by *coalesCgrip* [12] with a 15-min time-out.



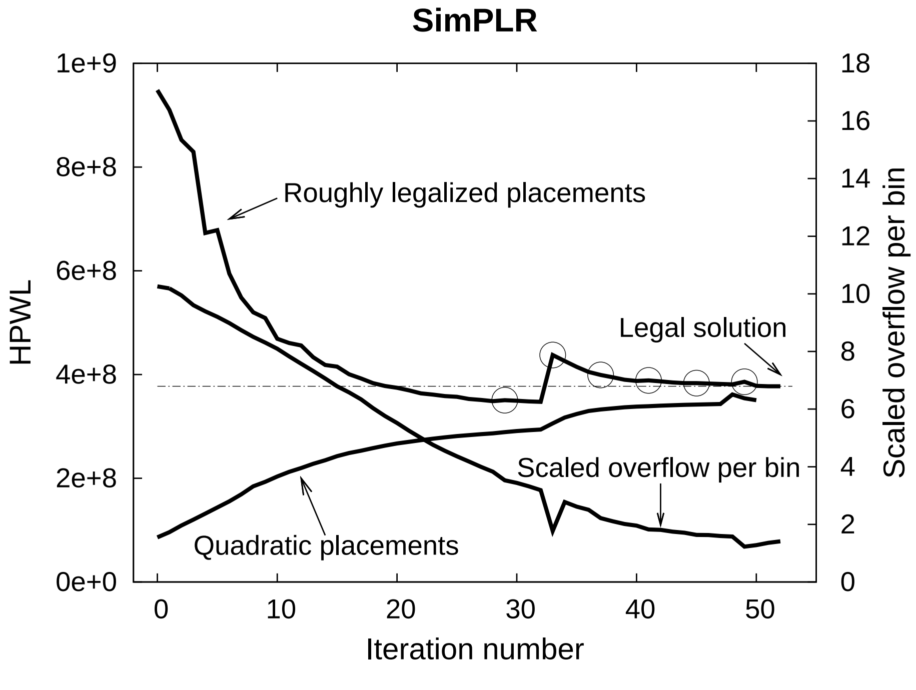


Figure 5.4: Progress of SimPL and SimPLR algorithms plotted against iteration counts (SUPERBLUE12). Each invocation of lookahead routing is marked with a circle. The second invocation of LAR and subsequent cell bloating visibly disrupt the quality of roughly legalized placements, with a smaller impact on quadratic placement.

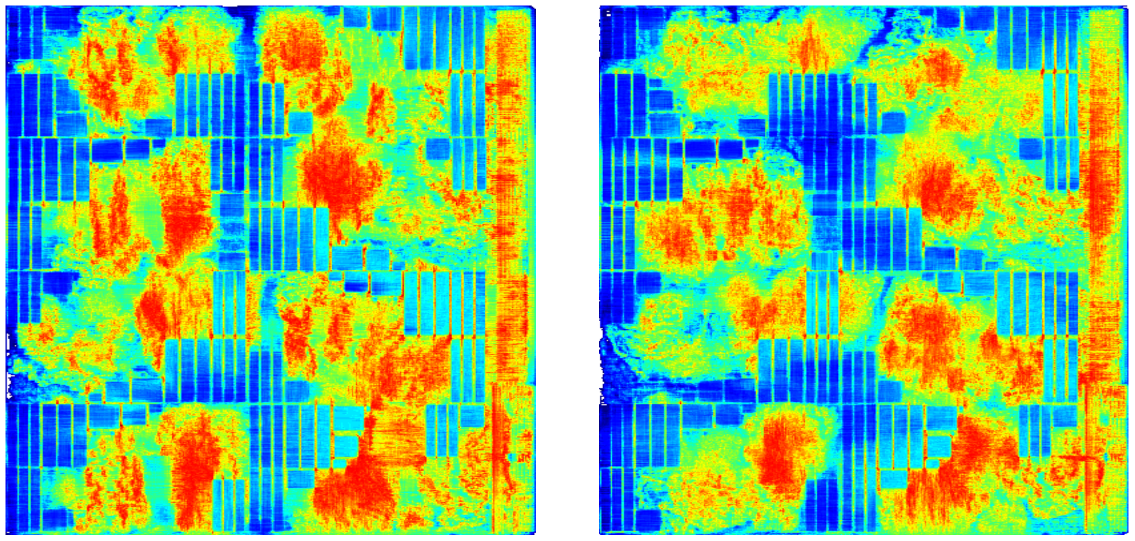


Figure 5.5: Congestion maps for SUPERBLUE15 for the best-reported placement at the ISPD 2011 contest (left) and SimPLR (right). Isolated red regions indicate peak congestion, dark-blue rectangles show unused resources.

router is used at its full strength.[5] *Thus, modern place-and-route leaves room for improvement in both gate locations and wire routes, but such improvements are best achieved in cooperation.* Such optimizations use physical resources more efficiently, enable smaller dies, and decrease IC manufacturing cost [95].

Given that SimPLR internally invokes a full-fledged router with a limited number of iterations (BFG-R) that produces a *valid* routing solution, better optimized results can be requested at the cost of greater runtime. The SimPLR framework can target specific nets and facilitates several further extensions, especially in timing optimization, where the placer's early and direct access to actual routes can improve the accuracy of delay estimation.

[5]The placement solutions produced by SimPLR on ISPD 2011 benchmarks, as well as resulting routes, are available on demand.

	Best in Contest				SimPLR with Ca-DP			
BENCHMARK	coalesCgrip **12hr**		coalesCgrip **24hr**		coalesCgrip **12hr**		coalesCgrip **24hr**	
	RtWL	OF	RtWL	OF	RtWL	OF	RtWL	OF
SUPERBLUE1	15.02	**0**	15.02	**0**	15.02	**0**	15.02	**0**
SUPERBLUE2	31.11	41408	31.20	16768	29.39	**39132**	29.52	**9646**
SUPERBLUE4	10.87	2466	10.88	1262	10.67	**44**	10.67	**36**
SUPERBLUE5	17.38	19112	17.49	10582	17.11	**9510**	17.18	**4392**
SUPERBLUE10	25.09	31320	25.16	**10652**	26.68	**22268**	26.73	11104
SUPERBLUE12	22.99	5130	23.02	594	23.00	**3302**	23.04	**0**
SUPERBLUE15	17.97	4448	17.98	2264	17.09	**0**	17.98	**0**
SUPERBLUE18	9.86	**0**	9.86	**0**	10.63	**0**	10.63	**0**
Average	**1.00**×	**19.41**×	**1.00**×	**8.43**×	**1.00**×	**4.12**×	**1.00**×	**1.00**×
Geometric mean	**1.00**×	**9.00**×	**1.00**×	**3.09**×	**1.00**×	**2.99**×	**1.00**×	**1.00**×

Table 5.3: Routed wirelength (RtWL, ×10e6) and routing overflow (OF) on ISPD 2011 benchmarks [108]. Routing was done using *coalesCgrip* [12] with a longer time-out than in Tables 5.1 and 5.2. Means are calculated excluding routable benchmarks, which under-represents the impact of proposed techniques.

5.6 Conclusions

Tight integration of major CAD tools is sometimes frowned upon in the industry because it may sharply increase software complexity, introduce subtle discrepancies and complicate software maintenance. However, such integration is highly sought in place-and-route, where high-performance global placers often generate hard-to-route solutions, creating unnecessary complications for downstream tools. The strategy pursued in our work is to give the placer advance, firsthand access to tentative net routes and resulting actual congestion maps (rather than crude estimates), as well as the ability to respond early and often. We believe that our proposed integration of global routing into global placement, *based on lookahead routing of upper-bound placements in the SimPL* algorithm, offers a particularly promising path to effective simultaneous place-and-route. Through a lightweight interface, the placer and the router quickly exchange multiple updates to cell locations and net routes, while maintaining the separate software infrastructure. Despite this software separation, the evolution of routes and cell placements are coupled and occur simultaneously. Empirical data show that our techniques improve routability, reducing total overflow compared to top results from the ISPD 2011 contest by $2.03\times$.

PART III

Scaling Global Routing to Larger Designs and Applications

CHAPTER VI

Taming the Complexity of Coordinated Place and Route

IC performance, power dissipation, volume, and signal integrity are currently dominated by interconnect effects. However, with ever-shrinking standard cells, blind minimization of interconnect length during placement causes routing failures. In this chapter, we develop Coordinated Place-and-Route (CoPR), consisting of (i) a Lightweight Incremental Routing Estimation (LIRE) frequently invoked during placement, and (ii) placement techniques that efficiently address multiple types of routing congestion. LIRE comprehends routing obstacles and non-uniform routing capacities, and relies on a new cache-friendly routing algorithm. On the ICCAD 2012 Benchmark suite, we outperform the winners of the ICCAD 2012 Contest.

6.1 Introduction

Throughout the history of EDA, Moore's law has been the primary driver for developing new, more scalable physical design techniques. In the 1990s, interconnect started lagging behind devices in power, delay and volume, making not only algorithmic scalability, but also the quality of optimization at large scale a key concern. In the next ten years, a variety of circuit phenomena, such as coupling capacitance and signal integrity, stipulated that *global routing* cannot be viewed as a standalone optimization. Interconnect stacks grew from three metal layers in the 1980s to 9-12 metal layers with non-uniform pitches today [109, 110], changing the nature of global routing algorithms (compared to channel routing) and increasing their impact on design quality. *Placement* is also no longer standalone, as it interacts with numerous other optimization steps to control interconnect lengths and delays. In particular, the idea of guiding global placement by routability estimation has been widely accepted for at least 15 years, used in commercial EDA tools, and promoted by academic (ISPD, DAC, ICCAD) Contests [108–110]. In such integrative research, understanding the strengths and weaknesses of existing optimizations becomes essential, as well as invoking the right primitive at the right time. Indeed, *complexity* — both the number of steps executed at runtime and the number of lines of code — is the main gating factor for what can be achieved by EDA tools in the foreseeable future.

In this chapter, we develop a streamlined system for Coordinated Place-and-Route (CoPR) that (i) uses cache-friendly routing primitives to quickly and accurately estimate routing congestion (LIRE), and (ii) offers a new categorization of congestion and new congestion-relief techniques during placement. These contributions are validated by strong empirical results on ICCAD 2012 contest benchmarks from IBM Research [110]. The remainder of this chapter is structured as follows. Section 6.2 presents our fast and accurate routing estimation technique. Section 6.3 introduces our placement techniques to proactively alleviate routing congestion. Section 6.4 describes the interactions between the placer and the routing estimator. Section 6.5 compares our techniques to currently-known approaches. Section 6.6 empirically validates the scalability of our techniques. Section 6.7 concludes our discussion.

6.2 LIRE: Routing Estimation

We develop a Lightweight Incremental Routing Estimator (LIRE) that quickly produces congestion maps as accurate as those by a global router (Figure 6.7). Empirically, we target 75K nets per second,[1] but also facilitate a tradeoff between quality and runtime. In contrast, modern routers [43, 77] complete 6K nets per second.[1]

Notation. We consider an $X \times Y$ routing grid $G(V, E)$ with (i) a set V of *GCells* (nodes) where each GCell $v \in V$ has integer coordinates (x_v, y_v), and (ii) and a set E of directed edges $e = (v_1, v_2)$, where the weight w_e of edge e encapsulates routing congestion and history costs (Lagrangian multipliers). Each node $v \in V$ is adjacent to its four cardinal neighbors: `NORTH` $(x_v, y_v + 1)$, `SOUTH` $(x_v, y_v - 1)$, `EAST` $(x_v + 1, y_v)$ and `WEST` $(x_v - 1, y_v)$. Consider a point-to-point connection π between two distinct GCells $S, T \in G$. When $x_S \leq x_T$ and $y_S \leq y_T$, a *forward* edge for π is an edge (v_1, v_2) such that $x_{v_1} < x_{v_1}$ or $y_{v_1} < y_{v_2}$, i.e., `EAST` or `NORTH`, and a *backward* edge for π is an edge (v_1, v_2) such that $x_{v_1} > x_{v_2}$ or $y_{v_1} > y_{v_2}$, i.e., `WEST` or `SOUTH`. Definitions for the three other orientations of π are symmetrical.

Key definitions. A *route segment* is a directed path in the routing grid. A *flat* route segment is a set of directed edges that are all `NORTH`, `SOUTH`, `EAST` or `WEST`. A *monotonic* segment is a connected set of flat segments such that each flat segment is either: (i) `NORTH` or `EAST`, (ii) `NORTH` or `WEST`, (iii) `SOUTH` or `EAST`, or (iv) `SOUTH` or `WEST`. Each monotonic segment is classified as `NORTH-EAST`, `NORTH-WEST`, `SOUTH-EAST`, or `SOUTH-WEST`. A *route* r_π is a collection of routing segments linking S and T.

[1]Median single-thread router performance on placements by the top three ICCAD 2012 contestants (Intel Xeon 3.4GHz CPU).

Algorithm 7 Bellman-Ford Algorithm with Non-negative Weights

```
Input:    Point-to-point connection $\pi$, Search Space ($V'$,$E'$)
Output:   route $r_\pi$

 1: for $i$ from $1 \rightarrow |V'|$ do
 2:     $cost[v_i]$ = $\infty$;
 3: end for
 4: $cost[S]$ = 0;
 5: for $i$ from $1 \rightarrow |V'|$ do
 6:     for $j = 1$ from $1 \rightarrow |E'|$ do
 7:         $e_j = (v_1, v_2)$;
 8:         if $cost[v_2] > cost[v_1]+$ COST($e_j$) then                ▷ relaxation
 9:             $cost[v_2] = cost[v_1]+$ COST($e_j$);
10:             $parent[v_2] = v_1$;
11:         end if
12:     end for
13: end for
14: $r_\pi$ = TRACE_PATH($\pi$);
```

6.2.1 Faster Routing

Global routing spends the majority of runtime finding shortest (weighted) paths in highly-congested regions. Such *maze routing* is necessary in congested regions for both *global routing* and *accurate congestion estimation* because, unlike pattern routing, it adequately captures detours. Detours are shaped by edge weights, which include congestion and history costs [43]. These weights must be maintained with sufficient accuracy and can be neither binned nor rounded without adverse impact on resulting routes. Therefore, shortest-path routing in congested regions is performed by A*-search. However, (i) the priority queue in A*-search is responsible for an extra $O(\log V)$ term in the overall complexity of the algorithm, (ii) priority queues, even when implemented using Fibonacci heaps, are too slow [69], whose pointer-based algorithms can experience costly cache misses, (iii) typical A* admissible functions based on straight-line distance become ineffective when history-based costs become large, and (iv) A*-search cannot leverage incrementality, i.e., given a candidate path, it cannot check optimality or perform an incremental improvement.

Linear-time cache-friendly routing. Given that A*-search is derived from Dijkstra's algorithm [30, Section 24.3], we hope to avoid these priority-queue-based approaches. Of the classic weighted shortest-path algorithms, the *Bellman-Ford (BF) algorithm* [30, Section 24.1] is array-based and moreover preserves memory locality. However, it may require V linear-time passes, taking $O(V^2)$ time. Notably, the *worst-case complexity* of Bellman-Ford (BF) can be avoided in global routing. Recall that each BF pass performs $E \times O(1)$ *relaxation* steps. When no relaxations in a pass result in improvement, no further improvement is achieved in later passes. Thus, BF can be terminated early without the loss of optimality.

During global routing, we consider one point-to-point connection ($S \to T$) at a time. Routing is limited to a subgrid $G'(V', E') \subseteq G$ enclosed in an isothetic (coaxial) bounding box that contains S and T. To generate a route, we visit the nodes of G' in a specified ordering $v_0, v_1, \ldots, v_{|V'|-1}$.[2] While the Bellman-Ford algorithm supports any node visitation ordering, we specify an ordering that not only affords us the highest memory locality, but also caters to the common case of monotonic paths. Starting from S, the nodes are traversed in the row containing S, toward T, and at each node v, relaxation is performed (lines 8-9 in Algorithm 7) along the four v-incident edges pointing toward T. The nodes in the next row closer to T are traversed, and so on until the row that contains T. As long as the node traversal follows the in-memory array layout (by rows or by columns), this method maintains the locality of memory access and is cache-friendly.

We propose to optimize BF passes with *duplex-edge relaxation*. At each edge considered by this technique, relaxations are attempted in both directions, but only *forward-looking edges* are considered at each vertex. While the same number of edges is considered per pass, cache utilization and memory locality are improved because for each

[2] Edges are traversed in the increasing order of adjacent vertices.

adjacent vertex (and edge cost) loaded from memory, two relaxations can be attempted rather than just one. Furthermore, if the first relaxation succeeds, the second one cannot occur — this saves an extra comparison. For example, at node v with coordinate (x, y), we relax either the incoming NORTH edge $(x, y) \rightarrow (x, y + 1)$ *or* the outgoing SOUTH edge $(x, y+1) \rightarrow (x, y)$. A similar duplex relaxation is performed in the EAST and WEST directions. By explicitly modeling via costs within these traversals [43, Section 3.4], BF will prefer fewer-bend routes.

Monotonic routing with one linear-time BF pass. As a special case, an optimal monotonic route can be found by (i) considering only forward edges (e.g., NORTH and EAST), and (ii) fixing the considered space to the bounding box b with dimensions $w \times h$, $w = x_T - x_S + 1$ and $h = y_T - y_S + 1$, that minimally contains S and T. Let t be the $w \times h$ matrix where $t[x][y]$ stores the partial cost from S with coordinates $(0, 0)$ to a node $v = (x, y)$. By construction, the cost at (x, y) depends solely on the costs at $(x - 1, y)$ and $(x, y - 1)$. Therefore, by visiting the nodes in row order (or column order) from S toward T, we visit every node in b exactly once. Since b has $w \times h$ nodes, the runtime complexity is $O(wh)$.

Non-monotonic routing with one linear-time BF pass. Recall that BF supports any node (and edge) ordering. Some optimal non-monotonic routes can be found in linear time within the bounding box b that minimally contains π by (i) employing duplex-edge relaxation and (ii) *echo-relaxation* if the relaxation succeeded in the direction opposite to node ordering (from a greater-numbered node to a smaller-numbered node). That is, in the forward-going node ordering, if a backward edge at node $v(x, y)$ results in a cheaper-cost route, we forward-propagate the cheaper cost through all recently-relaxed edges incident to v. Figure 6.1 illustrates finding an optimal route with three distinct monotonic segments in one BF pass. This improvement is effective in detouring short nets, and a majority of nets are short in practice. A more powerful variant of echo-relaxation would propagate costs through *all* incident edges, and allows BF to find longer detours (not used in this work).

Non-monotonic routing with BF and Yen's improvement. J. Y. Yen [125] suggested that *reversing* the node ordering between BF passes reduces the number of passes required to find an optimal path. We refer to the Bellman-Ford algorithm with early termination and Yen's improvement as BFY. Two and three BFY passes can quickly find long detours, as illustrated in Figure 6.2. This is especially applicable for large nets.

Theorem VI.1. *Let π be a point-to-point connection. Finding a minimal-cost route r_π^{min} with m (distinct) monotonic segments requires at most m BFY passes.*

Proof: Let t be the $w \times h$ matrix, where $t[x][y]$ stores the cost of the optimal path from its cardinal neighbors. Consider the first pass where partial costs are not yet propagated. By construction, $t[x][y]$ only depends on $t[x-1][y]$ and $t[x][y-1]$, and requires one BF(Y) pass. Therefore, an optimal route r_π with $m = 1$ monotonic segments is found after $m = 1$ passes. Consider the general case where r_π has k distinct monotonic segments. By assumption, r_π is formed using k BFY passes. By the early termination criterion, if BFY changes no costs in t, then $r_\pi = r_\pi^{min}$. If relaxation is successful during the backward pass, then r_π is allowed to *detour* through some intermediate node v' such that the route cost from S to v is reduced by going through v'. If such v' exists, then there exists a new path from S to v through v' such that the new path has an additional monotonic segment. During the forward pass, the full path of S to T through v. If going through v reduces the cost, then there is an additional monotonic segment $v \rightarrow T$. Therefore, for two additional BFY passes for an r_π with k distinct monotonic segments, we will generate a new path with $k + 2$ monotonic segments. Because we consider all intermediate nodes v as detours, the best-cost path will be stored. Therefore, if r_π^{min} has $m = k + 2$ monotonic segments, it will require $k + 2$ BFY passes. ■

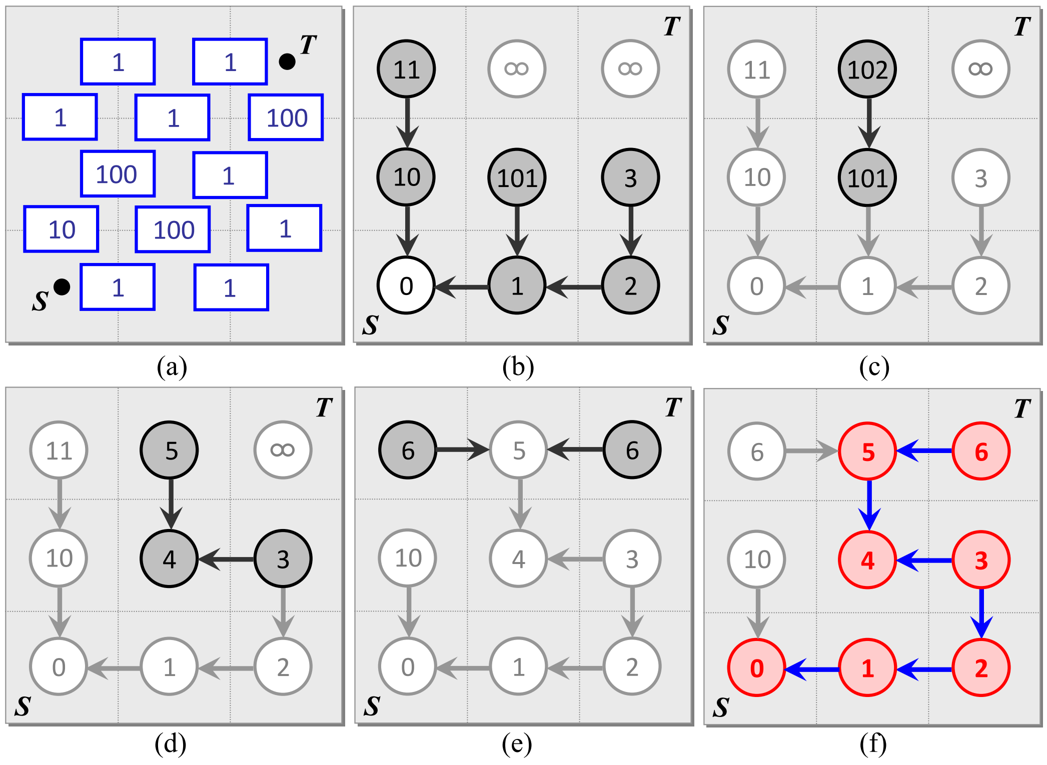


Figure 6.1: Applying one BF pass with duplex-edge relaxation and echo-relaxation to a point-to-point connection $S \rightarrow T$ without via-cost modeling. Arrows point to the previous node in the path. (a) The routing grid and edge costs (congestion). Let S have coordinate $(0, 0)$. (b) The partial costs of the first row and the center-left node have been populated. (c) Relaxing the `NORTH` $(1, 1) \rightarrow (1, 2)$ and `SOUTH` $(1, 2) \rightarrow (1, 1)$ edges at node with coordinate $(1, 1)$. (d) Relaxing the `EAST` $(1, 1) \rightarrow (2, 1)$ and `WEST` $(2, 1) \rightarrow (1, 1)$ edges at node with coordinate $(1, 1)$. The cost at $(1, 1)$ has been updated by the `WEST` edge and is propagated to $(1, 2)$. (e) The remaining nodes are considered, and partial costs are populated through T. (f) An optimal path with three monotonic segments is found in a single BF pass.

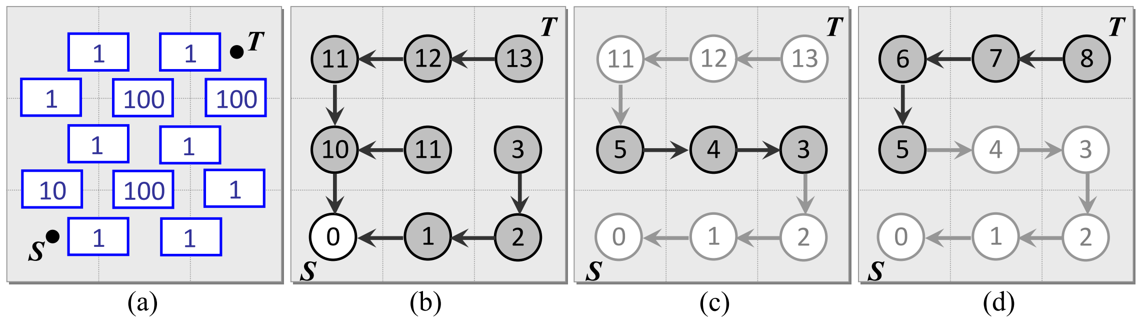


Figure 6.2: Applying BFY to a point-to-point connection $S \rightarrow T$ without via-cost modeling. (a) The routing grid and edge costs (congestion). (b) The first forward pass finds the optimal monotonic path of cost 13. (c) The backward pass finds a detour. (d) The second forward pass finds the optimal path of cost 8.

Theorem VI.1 is significant because in practice, many connections are routed with very few monotonic segments. In particular, most connections have very few bends [90], and the number of monotonic segments is upper-bounded by the number of bends. Furthermore, a route with many bends can still be monotonic (Figure 6.3b). In this context, Theorem VI.1 suggests that BFY typically finds shortest-path routes in $O(1)$ passes, and is therefore faster than A*-search (in addition to being cache-friendly). BFY runtime can be further reduced by limiting the number of passes with a small loss of optimality. For m relaxation passes in a bounding box of size $w \times h$, we need $O(mwh)$ runtime.

Incremental routing with BFY can use any existing route, including those previously found by A*-search. Instead of propagating the costs in a ∞-initialized table, we record the partial costs along an existing route (this is significantly faster than populating the entire table). Subsequent BFY passes find an optimal route, but require less runtime when a near-optimal initial route is available (Figure 6.3). Multiple such initial routes can be recorded in the BFY table before the first pass.[3] This type of incrementality speeds up not only rip-up-and-reroute and negotiated-congestion methods, but also repeated invoca-

[3]A speed-up common for A*-search limits search to GCells to narrow corridors around known routes. This significantly improves runtime for large nets, but may overlook shorter paths.

tions of LIRE during placement (Section 6.4 also outlines other types of incrementality supported by LIRE).

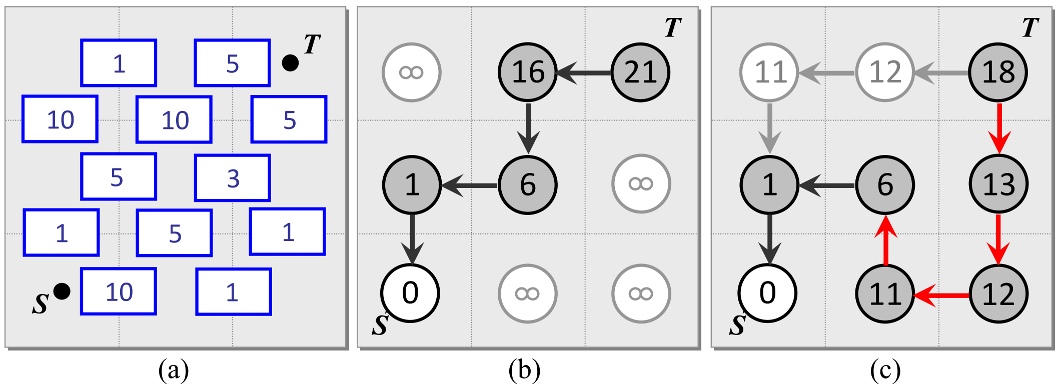


Figure 6.3: Applying BFY to an initial route for a point-to-point connection $S \rightarrow T$. (a) The routing grid and edge costs (congestion). (b) The initial route with cost 21. (c) Through relaxation, BFY can preserve part of the route, and find a better partial segment, resulting in a new route with cost 18.

Coarse-grid routing is based on the observation that large nets often admit near-optimal routes with long flat segments. Therefore, we reduce the search space by only considering every i^{th} row and j^{th} column. This allows us to find a reasonable-cost route quickly, and then incrementally relax it on a finer subgrid.

6.2.2 Fast and Accurate Estimation

Unlike true global routing, constructive congestion estimation needs not to optimize routes in congestion-free regions. Finding routes that avoid congested GCells is sufficient. Therefore, existing methods first evaluate several pattern routes (L, C, Z) and invoke more sophisticated algorithms only when needed. For similar reasons, eligible GCells are initially limited to the bounding box of the connection, which is gradually expanded until an acceptable route is found. LIRE too estimates congestion by catering to the common case first. For each point-to-point connection $\pi = S \rightarrow T$, LIRE limits the search space to the bounding box b with size $w \times h$ that minimally contains π. It considers the two L

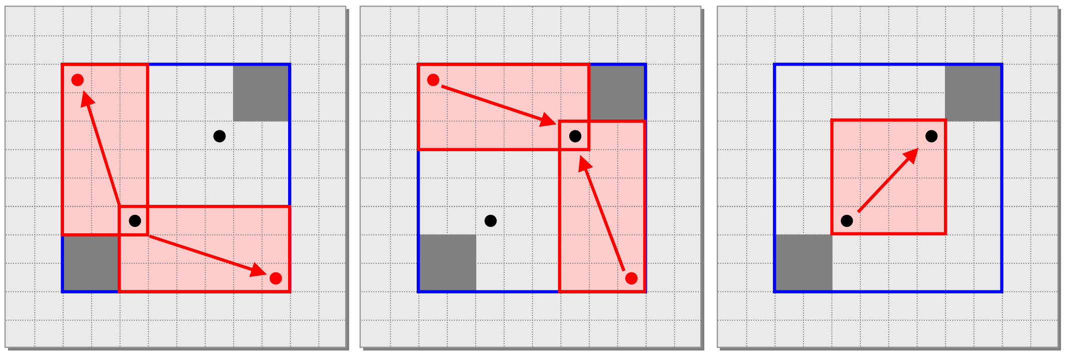

monotonic passes (*i*) and (*ii*)
monotonic passes (*iii*) and (*iv*)
monotonic pass (*v*)

Figure 6.4: Non-monotonic routing using the Bellman-Ford Algorithm with an expanded bounding box. The red arrows represent monotonic passes.

routes, with preference for congestion-free routes. If both routes are congested, BFY finds a route with $O(1)$ monotonic segments. If this route is congested, LIRE expands b to to $W \times H$, $w < W \leq X, h < H \leq Y$, which may be based on congestion [78].

Within this expanded rectangular bounding box, we only consider a partial rectilinear bounding box for two reasons. First, we can reduce the overall runtime by limiting the bounding box. Second, we observe that the space omitted only contributes to routes that require multiple detours. Since we do not (and need to) generate routes that heavily detour, we omit this search space to minimize detours. Within the rectilinear bounding box B, let UL and BR be B's respective upper left and bottom right corners. We then perform five monotonic-routing passes: (i) $S \rightarrow UL$, (ii) $S \rightarrow BR$, (iii) $UL \rightarrow T$, (iv) $BR \rightarrow T$, and (v) $S \rightarrow T$. While we can relax each monotonic pass, we have found that just expanding the original bounding box is sufficient. Our implementation expands bounding boxes up to twice the original size, but only for routes with extreme aspect ratios.

6.3 Congestion Relief

The main precept of routability-driven placement is to increase the *porosity* of placement regions with high routing congestion. Regardless of how congestion is estimated, porosity has traditionally been increased in two ways: (i) *after global placement*, by shifting cell locations [95, 126] and using congestion-driven detailed placement [38, 48, 65], and (ii) *during global placement*, by inflating cells based on early congestion estimates and pin density [38,48,65].

While studying the impact of these techniques on challenging IC layouts, we observed their insufficiency. Modifications performed after the global-placement phase must preserve the structure of resulting placement or risk unbearable deterioration of interconnect length. Cell inflation performed during global placement is more flexible and powerful. However, when inflated cells move outside the congested region, new cells must be inflated, and this process may consume all available whitespace without addressing the root cause of congestion in a given region (this phenomenon was confirmed to us by several industrial colleagues and academic colleagues). Further analysis revealed two previously unknown types of routing congestion, which we include below as types 2 and 3.

1) *cell-based* congestion is caused by cell-to-cell proximity,
2) *local layout-based* congestion is caused by static design properties, such as blockages and reduced routing capacities,
3) *remote-induced layout-based* congestion is attributed to non-local factors, such as long nets.

These congestion types are illustrated in Figure 6.5. The distinctions among them be blurred by inaccurate congestion maps and also during congestion reduction *after* global placement [95, 126] which does not drastically change cell locations. However, they become essential when guiding global-placement iterations by accurate congestion maps. Conceptually, type-2 congestion requires whitespace injection into relevant regions *in such a way that whitespace remains in these regions even when cells relocate*.

Cell-based congestion. As the placer spreads cells, it often implicitly keeps cells close together to decrease HPWL. However, this "clumping" creates difficult-to-route regions, as there may be too few tracks to accommodate all incident nets. This type of congestion is easily mitigated through cell inflation. However, inflating too many cells *or* inflating some cells by too much can exhaust whitespace too soon, inhibit convergence and undermine quality. To ensure steady improvement, we inflate each cell in the top 5% most congested GCells by computing its new width as follows.

$$\max\{\mathrm{width}(cell) + 1, 1 + \theta(G) \cdot \Lambda(cell) \cdot \deg(cell)\} \tag{6.1}$$

Here, $cell$ is a movable cell in a congested GCell, $\mathrm{width}(cell)$ and $\deg(cell)$ are the width and connectivity of $cell$, respectively. $\theta(G)$ is an adaptive function (described below) of the routing grid G, and $\Lambda(cell)$ is the number of times $cell$ has been in a congested GCell. We define θ similarly to [65, Equation 12], except that we upper-bound θ to limit how much a cell can be inflated.

$$\theta = \min\{0.5, \max\{0, \alpha \cdot \eta(G) \cdot \xi(G) + \beta\}\} \tag{6.2}$$

Here, $\eta(G)$ and $\xi(G)$ represent the respective *difficulty* and *routability* of the design, where $\eta(G)$ is the sum of every GCell congestion in G, and $\xi(G)$ is the ratio of the total GCell congestion in G. α and β are constants based on linear regression. Unlike previous cell-inflation approaches [65], our formula *does not include* the GCell's congestion. By excluding the numeric congestion value, we only rely on the routing estimator's accuracy for congestion *locations*, and less on the reported congestion value. This prevents excessive inflation, and facilitates a smooth placement transition.

Layout-based congestion. During HPWL-driven placement, the target density is often high, as this facilitates low-HPWL placement solutions. However, if the placer is not congestion-aware, it may pack cells in regions of high congestion. To this end, we seek

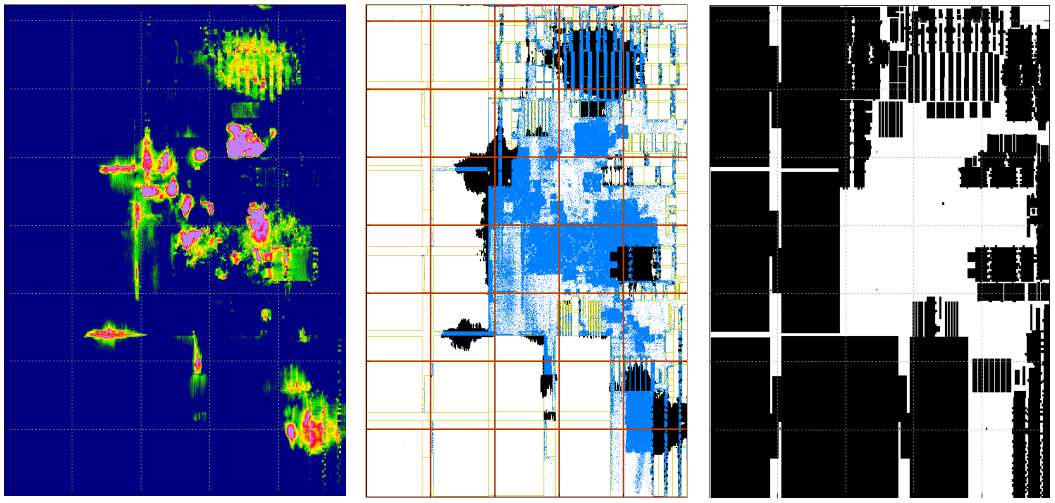


Figure 6.5: Congestion map produced after one BFG-R [43] iteration (left), placement map of cell locations (center), and blockages (right) for SUPERBLUE2 [109]. In the center, blue indicates movable cells, and black indicates congested GCells over blockages. Congestion is present around blockages (layout-based) and blockage-free regions (cell-based).

to locally increase whitespace to encourage cells to spread elsewhere. However, analytic placement frameworks are not always amenable to techniques that change (local) target density. Instead, we enforce non-uniform target densities in localized regions. We distinguish layout-based congestion as either *local*, which is caused primarily by static constraints such as custom routing-edge capacity reductions, or *remotely-induce*, where congested GCells contain no standard cells but have few routing tracks traversed by long nets. While the former can be addressed through locally injecting whitespace, the latter cannot, as there are no cells to move out of the congested region. In the remainder of this section, we discuss our method of enforcing non-uniform target density by (i) creating a *packing peanut* (fixed cell) at the center of every GCell, and (ii) modifying its size based on congestion.

Implementation. To address *local layout-based congestion*, we modify the size of packing peanuts during the initial HPWL optimization stage based on *pin density*, and during the global placement stage based on routing congestion. During initial placement, we coarsely estimate routing congestion of the design based on available routing capacity and cell pin density. We first divide the layout into 8×8 GCell regions and compute the number of pins in each region. We then (pessimistically) estimate that each pin in the region will occupy two routing tracks, and increase the packing peanuts' size based on the ratio of estimated usage and routing capacity. This approach of coarsely dividing the layout gives the placer a high-level outlook and encourages cells spreading to regions of lower pin density. We define two parameters: (i) the maximum expandable area $PA(g)_{max}$, which is based on the surrounding non-overlapping GCell areas, and (ii) the minimum area $PA(g)_{min}$, which is based on GCell pin density. Let $C(g)^k$ be the congestion of GCell g at routing iteration k. Then the packing peanut area $PA(g)$ at g is

$$PA(g) + 0.15 \cdot \big(PA(g)_{max} - PA(g)\big) \tag{6.3}$$

if $C(g)^k > C(g)^{k-1}$ and $C(g)^k > 1$. If the congestion is reduced but not removed, i.e., $C(g)^{k-1} > C(g)^k > 1$, then the packing peanut size remains the same. Otherwise, if congestion is removed, the size is reduced by 40%.

To address *remotely-induce layout-based congestion*, we increase the packing peanuts in GCells *closest to the congested region* and their neighboring GCells. Such modifications often occur around placement blockages. Across placement iterations, the packing peanuts increase placement porosity by reducing the demand in regions without blockages as well as customizing the resource distribution around blockages. Unlike rectangular macro expansion [48], our approach affords the placer a higher degree of flexibility as to where long nets can be shifted (Figure 6.9). To complement cell-inflation techniques, our approach can prevent allocated whitespace from moving away.

6.4 Coordinated Place and Route

The integration of routing estimation within placement allows us to leverage the existing infrastructure and avoids task redundancy. Giving LIRE up-to-date access to cell locations simplifies the construction of new congestion maps when placement changes.

Incremental placement updates. After its first invocation, LIRE maintains the overall congestion map and keeps track of the GCells traversed by each point-to-point connection π. At subsequent LIRE invocations, if the endpoints of π remain in the same GCells (despite changes in their continuous-valued locations), π's route and its contribution to the congestion map are left unchanged. While this type of incrementality has limited use in early placement iterations, its effect is more pronounced in later iterations and during detailed placement, when the placement has stabilized.

Incremental route updates. When invoked for the first time, LIRE generates routes from scratch. Subsequently, it tries to reuse existing routes where possible. Nets whose terminals relocated to different GCells are rerouted using the original net ordering, as outlined in Section 6.2. For remaining nets, we check if their routes are congested. Congestion is mitigated by BFY passes, and the bounding box is expanded if necessary. Incremental routing with BFY allows us to replicate the accuracy of a maze router while reducing runtime by (i) avoiding routing nets that have not changed and (ii) reusing (partial) routes.

Placement-routing interface for coordinated place-and-route:

- `LIRE::Initialize()` reads in the benchmark information, sets up the routing environment, and computes the *static* routing-edge capacities (e.g., due to blockages or custom capacity reductions). Dynamic capacity adjustments such as pin blockages in Section 6.6, are accounted for by `LIRE::updatePlacement()`.

- `LIRE::updatePlacement()` restructures the nets based on any placement changes, and maintains lists of nets that require full modification, as well as those that can be reused. Dynamic routing capacities are adjusted due to cell-location updates.

- `LIRE::route()` generates routes on an as-needed (lazy) basis. It decomposes each multi-pin net into two-pin subnets based on its MST, and follows the protocol outlined in Section 6.2.

- `LIRE::genCongMap()` translates edge capacities and usages to a GCell-centric congestion map as in [65], where a GCell is congested if any surrounding edge is congested.

The handling of design hierarchy is entrusted to the placer and does not add complexity to the place-and-route interface (Figure 6.6). In summary, we advocate a *coordinated* integration style of physical optimizations, where each component uses algorithms that are independently-meaningful and independently-efficient, but also are capable of taking external suggestions. Unlike *simultaneous place-and-route* advocated in [65], this type of integration limits software complexity, allows for component replacement and unit testing. It eases the integration of timing analysis and other components necessary for effective timing closure of modern SoC designs [69].

6.5 Comparisons to Prior Work

Comparing our techniques to prior art, we consider (i) point-to-point routing algorithms, (ii) using global routes versus probabilistic congestion maps, (iii) incremental routing techniques, and (iv) handling congestion around blockages.

Fast routing. The closest recent publication to our material in Section 6.2 is [78]. It also advocates replacing A*-search with fast linear-time routing algorithms that exploit monotony (although our work was completed before [78] was published or available to us).

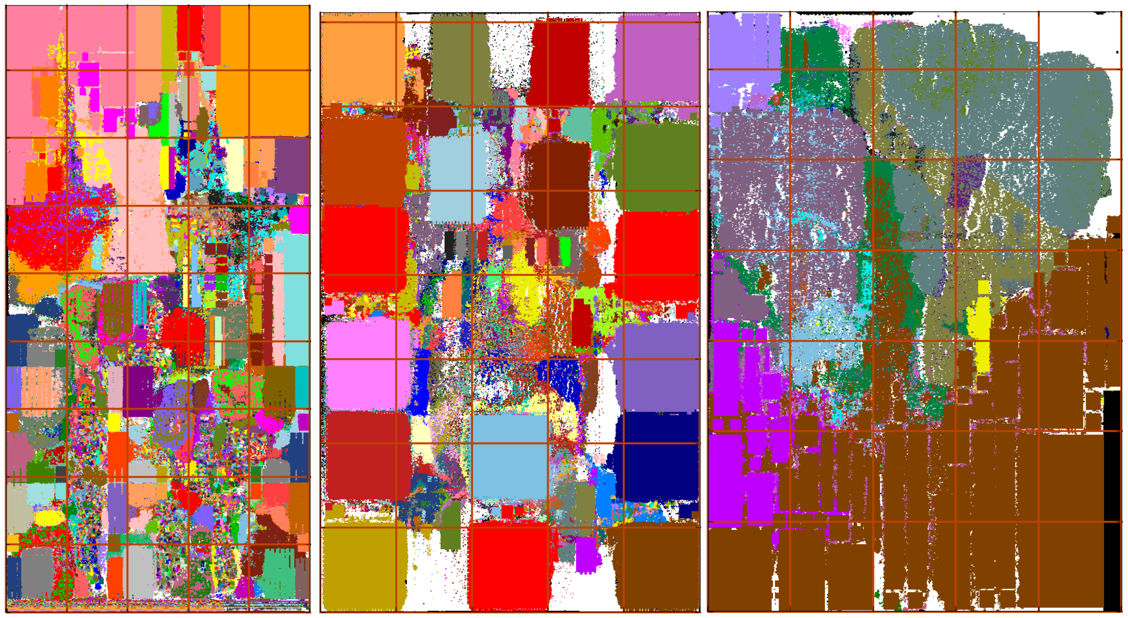


Figure 6.6: CoPR placements of the SUPERBLUE7 (left), SUPERBLUE10 (center), and SUPERBLUE18 (right) testcases [110].

However, their notion of monotonic routes is different, no optimality results are claimed, and CPU-cache effects are not considered. In terms of theoretical contributions, the connection we establish to the Bellman-Ford algorithm with Yen's improvement also appears new. Empirically, the RCE estimator [78] is not used to drive a competitive global placer, whereas we report successful results for coordinated place-and-route using LIRE. We believe that congestion-driven bounding-box expansion pioneered in RCE can be valuable, but have not had the time to implement and evaluate it.

The only modern description of an industry router that we could find is in [69]. It concedes that Dijkstra's algorithm [30, Section 24.3] (from which A*-search is derived) is "much too slow" for large modern netlists, even with Fibonacci heaps. However, rather than replace Dijkstra with linear-time algorithms as we do, the authors speed it up with sophisticated data structures (interval-based route-cost representations) and algorithms (sharper admissible functions for A*-search based on landmarks). Direct comparisons would be difficult to make, even if we had access to their source code, because advanced data structures use more memory and require significant up-front set-up, along with maintenance. However, a single-threaded version of LIRE takes only $<15\%$ of runtime in our entire place-and-route flow, despite frequent (>10) invocations by the placer (Table 6.4). Speeding it up further would have limited impact. Most importantly, we have advanced the goal of our research — to tame the complexity of place-and-route — by entirely avoiding sophisticated routing algorithms and data structures.

Congestion estimation must accurately identify hotspots and guide the placer to relieve congestion. While *probabilistic congestion maps* are easy to implement, they can be slower per net than constructive routing, as shown in [118]. They are also highly inaccurate, as recently articulated by IBM researchers [74]. Nevertheless, most routability-driven placers [38, 48] still use probabilistic methods. As depicted in Figure 6.7, a congestion map constructed using LZ-routing [90] differs from a router-based map. Figure 6.8 and Table 6.1 compares total overflow (TOF) between L-routing, LZ-routing, LIRE, and maze routing [43]. On average, LIRE overestimates TOF by 4% with no significant outliers.

Incremental routing techniques. All modern routability-driven placers [38, 48, 65] use built-in congestion estimation to construct new estimates from scratch on every invocation. This process is unnecessarily time-consuming, especially when the placement has not changed significantly. While some prior techniques rip-up and reroute some congested nets [127], they assume a static routing (and placement) instance. In contrast, our incremental techniques account for dynamic placement (and routing) instances, take advantage of previous (partial) routes, and update routes on an as-needed basis. These techniques are especially applicable to congestion estimators based on constructive global routing, but also should be helpful in full-fledged routers. Empirically, we matched the accuracy of a full global router with limited runtime overhead.

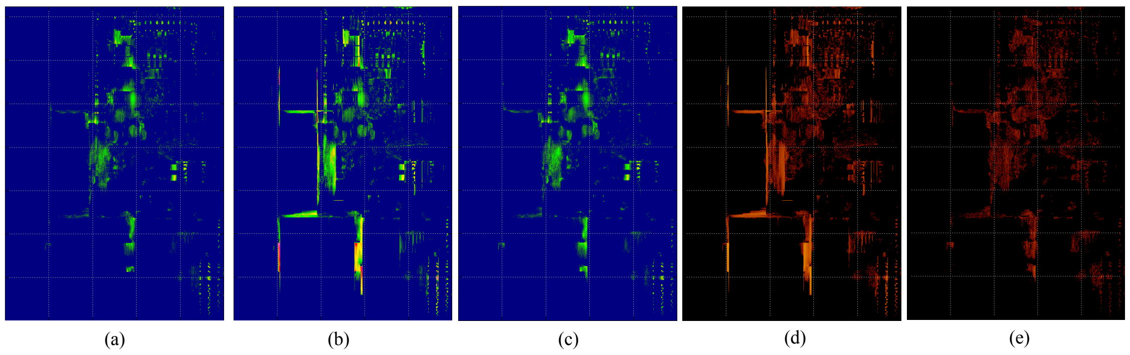


Figure 6.7: Comparison of routing estimation techniques on the SUPERBLUE2 benchmark [109]. The congestion map in (a) is produced by one iteration of BFG-R [43], in (b) — by LZ-routing, and in (c) — by LIRE. Images in (d) and (e) show how well (b) and (c) match (a) — ratios of congestion values are plotted. Orange indicate large differences and black — no difference. While all techniques overestimate congestion, LZ-routing and L-routing produce many false positives, whereas LIRE does not.

Placement and routing blockages, e.g., macro blocks, often lead to congestion around their borders. Previous work [48] proactively reserves resources by expanding macros. However, (rectangular) macro inflation is rather crude in controlling whitespace — it either allows all cells or prevents all cells in a given rectangular region. Our non-uniform target density, as implemented with packing peanuts, provides much more flexible control of whitespace, as shown in Figure 6.9. By increasing the packing peanut sizes in areas of

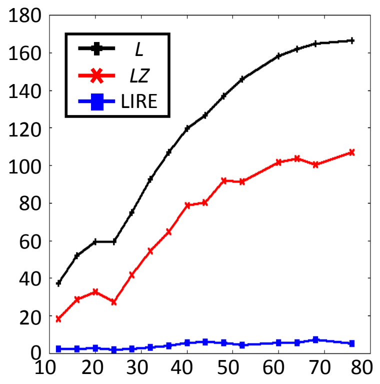


Figure 6.8: The error percentage of total overflow for L-routing, LZ-routing, and LIRE relative to (a) over the placement iterations of CoPR.

Iter.	Total overflow (e5)				Comparison vs. maze		
#	maze	L	LZ	LIRE	L	LZ	LIRE
12	31.04	42.59	36.78	31.80	1.372	1.185	1.024
16	20.41	31.00	26.30	20.91	1.519	1.289	1.024
20	16.00	25.49	21.22	16.45	1.594	1.327	1.039
24	15.13	24.13	19.31	15.13	1.595	1.276	1.020
28	11.68	20.44	16.58	11.96	1.749	1.420	1.024
32	7.880	15.17	12.16	8.149	1.925	1.544	1.034
36	6.424	13.29	10.59	6.684	2.069	1.649	1.041
40	5.452	11.99	9.745	5.755	2.199	1.787	1.056
44	5.051	11.44	9.108	5.359	2.266	1.803	1.061
48	4.636	10.98	8.895	4.898	2.369	1.919	1.057
52	4.375	10.75	8.382	4.575	2.458	1.916	1.046
60	3.825	9.876	7.721	4.043	2.582	2.019	1.057
64	3.718	9.736	7.572	3.931	2.618	2.036	1.057
68	3.697	9.796	7.410	3.964	2.650	2.004	1.072
76	3.503	9.337	7.254	3.684	2.665	2.071	1.052
Avg					2.06×	1.65×	1.04×

Table 6.1: Total overflow estimation comparisons of L-routing, LZ-routing, the initial (maze) routing of BFG-R [43], and LIRE inside CoPR for the SUPERBLUE2 benchmark [109] (Figure 6.8).

congestion *and* in selected neighboring GCells, we allow cells to move into congestion-free regions around macro borders, whatever shape those regions may assume.

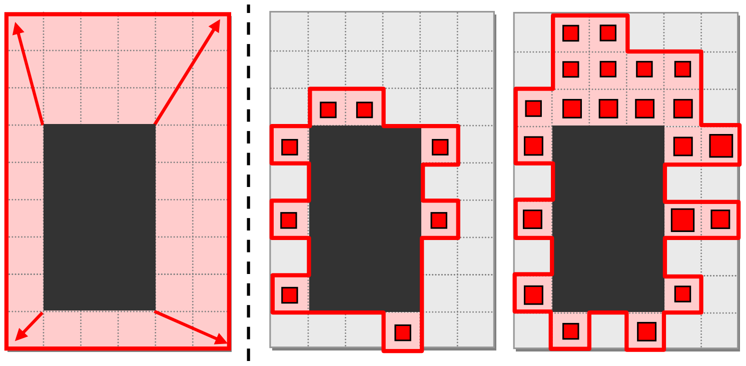


Figure 6.9: Congestion-driven rectangular macro expansion [48] (left) versus our technique (right).

			Quality metrics using NCTUgr [77] (e8)			
Benchmark	Nodes	Nets	SimPLR (1)	Ripple (2)	NTUplace4 (3)	CoPR
SUPERBLUE1	847K	822K	2.789	2.889	2.850	2.849
SUPERBLUE3	920K	898K	3.439	3.604	4.477	3.401
SUPERBLUE4	600K	567K	2.434	2.269	2.360	2.343
SUPERBLUE5	772K	787K	3.603	3.486	4.217	3.556
SUPERBLUE7	1.36M	1.34M	4.313	4.291	4.137	4.379
SUPERBLUE10	1.20M	1.15M	6.909	6.111	7.190	6.706
SUPERBLUE16	699K	697K	2.857	2.840	2.833	2.804
SUPERBLUE18	1.27M	469K	1.823	1.791	1.709	1.699
Ratios of averages (×)			1.02×	1.00×	1.06×	1.00×

Table 6.2: Quality metrics (based on NCTUgr [77]) *without runtime* for the top three contestants as reported at the ICCAD 2012 Routability-driven Placement Contest [110]. Full results for SimPLR, RippleCUHK and NTUplace4h are available at [110].

			Quality metrics using BFG-R [43] (e8)			
Benchmark	Nodes	Nets	SimPLR (1)	Ripple (2)	NTUplace4 (3)	CoPR
SUPERBLUE1	847K	822K	3.023	3.341	2.962	3.004
SUPERBLUE3	920K	898K	3.803	3.906	4.609	3.801
SUPERBLUE4	600K	567K	2.865	2.659	2.773	2.501
SUPERBLUE5	772K	787K	3.980	3.654	3.919	3.835
SUPERBLUE7	1.36M	1.34M	4.479	4.502	4.283	4.503
SUPERBLUE10	1.20M	1.15M	8.114	7.080	7.810	7.561
SUPERBLUE16	699K	697K	3.117	2.929	3.032	3.097
SUPERBLUE18	1.27M	469K	2.461	2.207	1.838	2.228
Ratios of averages (×)			1.05×	1.00×	1.01×	1.00×

Table 6.3: Quality metrics (based on BFG-R [43]) *without runtime* for the top three contestants as reported at the ICCAD 2012 Routability-driven Placement Contest [110] and CoPR. Full results for SimPLR, RippleCUHK and NTUplace4h are available at [110].

6.6 Empirical Validation

Our algorithms are implemented in C++ in a tool called CoPR (pronounced "copper") using the OpenMP library [32] and compiled with g++ 4.7.0. Our global placer was derived from SimPL [67], which was the case for three out of the top four teams at the ICCAD 2012 Contest [110]. Thus, the choice of the global placement algorithm was not a significant factor in relative performance.

Empirical results are reported on the ICCAD 2012 benchmark suite [110] derived by

Benchmark	Total (s)	Ratio vs. SimPLR	LIRE calls	LIRE %
SUPERBLUE1	1047	1.042	14	13.1%
SUPERBLUE3	1248	0.926	14	15.2%
SUPERBLUE4	902	1.431	21	14.4%
SUPERBLUE5	1084	0.966	12	12.4%
SUPERBLUE7	1796	1.072	12	11.2%
SUPERBLUE10	2585	1.048	21	17.3%
SUPERBLUE16	625	0.711	12	15.4%
SUPERBLUE18	692	0.925	18	19.6%
Average		1.00×		14.6%

Table 6.4: CoPR runtimes are compared to those of the fastest top-3 contestant SimPLR by running both tools on the same server (3.4GHz Intel Xeon). The last two columns show the runtime of LIRE as a percent of total CoPR runtime, and the number of LIRE invocations on each benchmark.

IBM researchers from industry designs. Some of these benchmarks were released only after the results of the ICCAD 2012 Contest were announced. The overall figure of merit combines quality metrics (interconnect length, routing congestion evaluated by a router, and pin blockage) and runtime. Tables 6.2 and 6.3 compares CoPR to official contest results [110] for the top three contestants. In terms of quality metrics based on the NCTUgr router (without runtime) in Table 6.2, CoPR outperforms NTUplace4h by 6% and SimPLR by 2%, and matches the overall quality of Ripple, which is 5.7× slower. In terms of quality metrics based on the BFG-R router (without runtime) in Table 6.3, CoPR outperforms NTUplace4h by 1% and SimPLR by 5%, and matches the quality of Ripple. Table 6.4 compares CoPR's runtime. Its runtime regime intentionally matches that of SimPLR (the fastest top-3 contestant), which trails CoPR in quality. The last two columns show that LIRE is called 12-21 times by CoPR per benchmark, and yet uses <15% of CoPR's runtime in total.

6.7 Conclusions

Our work deals with an alarming trend in the design or digital random-logic blocks, where interconnect's dominance in area, volume, delay, power and signal-integrity is increasing with every new technology node [52]. If unchecked, this trend is threatening to render Moore's law irrelevant — packing more devices on a chip is useless if they cannot be effectively connected. The most direct and effective remedy known today is to reduce interconnect demand, which can be done by optimizing standard-cell locations and wire routes. As articulated recently by IBM researchers, design flows with separate placement and routing steps have become ineffective for modern ICs [109], but combining the two brings tangible and significant benefits in IC cost [95]. However most of physical-design research continues focusing on standalone optimizations, partly due to the complexities involved in place-and-route integration. These complexities include sophisticated data structures and elaborate multistep optimizations used by state-of-the-art algorithms [69], unmaintainable source-code bases that are unnecessarily entangled, large sets of tuning parameters that may need to be adjusted to individual benchmarks, and of course significant runtime. In this work, we develop an algorithmic framework for coordinated place-and-route (CoPR) that combines independently-meaningful components and systematically reduces the complexities of place-and-route. Our contributions fall into three categories: (i) dramatic acceleration of constructive routing estimation through linear-time cache-friendly algorithms that do not require sophisticated data structures, (ii) significant reductions in the amount of work through pervasive incrementality at the interface between placement and routing, (iii) identification of two new types of routing congestion, as well as mechanisms by which a placer can diagnose them and respond effectively, and (iv) strong empirical results on recent benchmarks from IBM Research.

Our results will lead to more compact (less costly) IC layouts, along the lines of experiments in [95], as well as much faster back-end turn-around-time that would allow IC designers to evaluate a greater number of micro-architectural configurations.

CHAPTER VII

Addressing the Buffer-explosion Problem Through Low-cost Heterogeneous 3D Integration

We now employ the use of a global router within a chip-design flow to address the *buffer explosion problem*, i.e., where long interconnects in modern ICs require an increasing large number of large buffers due to CMOS scaling. Seeking to improve layout efficiency, we observe that large buffers do not need state-of-the-art technology, and can be segregated onto an older-generation, low-cost *buffer-die* that is stacked below the main *logic-die*. Such heterogeneous 3D integration not only saves area on the more expensive die, but also alleviates routing congestion by removing via-stacks to high-metal layers. With moderate buffer-die overhead, our technique decreases routing congestion and cost.

7.1 Introduction

Interconnect-related physical design challenges are caused by technology scaling and the continued growth of netlists. Since the 180nm technology node, interconnect delay (e.g., RC delay) has dominated chip-level performance. As this RC delay grows quadratically with respect to wirelength scaling, buffers are inserted to maintain linear delay [85], where the rate of growth is based on wirelength increase and technology scaling[1]. How-

[1]Under ideal geometric scaling, the total wirelength for a given die size increases by $1.4\times$ per node, and the critical repeater distance decreases by $1.67\times$ per node [6].

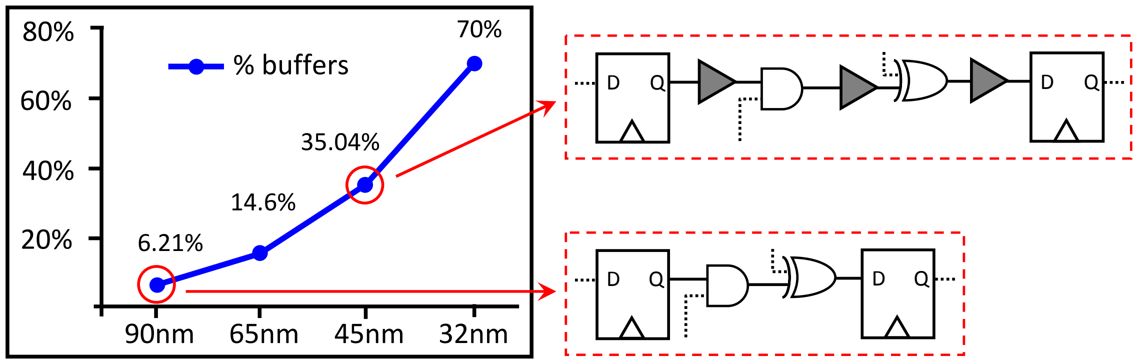


Figure 7.1: Buffer explosion with technology scaling [97].

ever, inserting buffers (i) increases silicon area, which forces cells to spread farther apart, and (ii) scales up interconnects, which in turn requires more buffers to compensate for the additional delay. This vicious cycle creates the *buffer explosion* problem, as predicted by Saxena et al. [97] and illustrated in Figure 7.1.[2]

In VLSI physical design, the routing stage is the most sensitive to design changes. For example, if there are insufficient resources, routing will be time-consuming and prone to failure, often requiring a loop back to placement, synthesis or earlier design stages. Increased design complexity (2× per technology node) also increases the difficulty of achieving a legal routing solution. Long interconnects routed on high metal layers will consume many vias when buffered; the via stacks cause routing blockage that can force neighboring wires to detour [60], as illustrated in Figure 7.2. Detours cause blockage, which in turn cause further detouring; this is another vicious cycle which degrades routability and challenges the router's ability to complete the layout without violations.

Previous research addresses the challenges of buffer explosion and routability chiefly through physical design optimizations that seek to reduce total wirelength and hence improve routability. Recent routability- and wirelength-driven academic placement and rout-

[2]Saxena et al. [97] predicted that at the 32nm technology node, over 70% of instances would become buffers or inverters (i.e., repeaters). However, serial vs. parallel tradeoffs in logic structure can modulate the number of long global interconnects and buffers, possibly at the cost of chip performance.

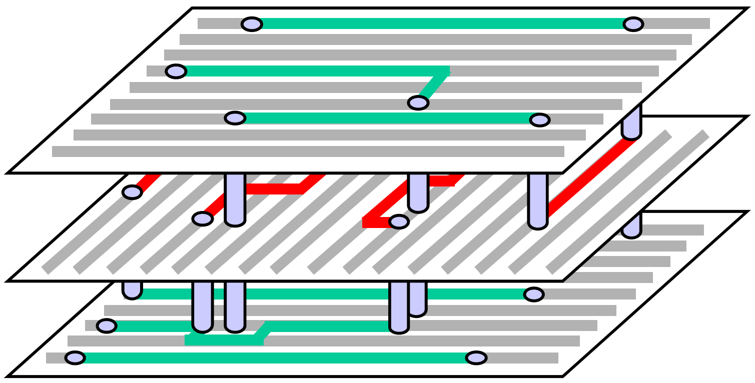


Figure 7.2: Wire detouring due to via blockage.

ing research include SimPLR [65], BFG-R [43], etc. However, the design space for a 2D, single-die layout is restricted by the die area, which is limited by constraints on yield, cost, power, and other parameters. As has been widely noted in recent years, 3D integrated circuits offer significant performance benefits over 2D integrated circuits, mainly by reducing the length of interconnects. TSV-based [99] and monolithic [8] 3D IC integration can vertically connect stacked transistors, enabling reduction of average interconnect length in the physical implementation. Previous research efforts mainly focus on 3D IC partitioning [44], floorplanning [19], placement [27] and routing [29]. Heat dissipation is a major challenge on 3D IC due to the rapid increase of power density with die stacking and the low thermal conductivity of dielectric layers sandwiched between adjacent dies. This has motivated research on 3D IC thermal planning, e.g., [28]. To maintain convergence and a manageable number of iterations in the physical implementation flow, further innovations are required to mitigate the buffer explosion and routing congestion challenges in leading-edge chip implementation.

In this work, we propose a *heterogeneous 3D* (hetero-3D) solution to address the buffer explosion and routability problems. Recall that buffers are typically inserted into long wires, which tend to be routed on top metal layers. However, buffers are located below Metal1 and can only be connected to top metal layers by a series of vias. These vias can obstruct intersected routing tracks, aggravating routing congestion on all metal layers.

Our **first** insight is that by positioning buffers *above* the top metal layers, we can significantly reduce demand for vias and increase the number of available routing tracks. Our **second** insight is that we can accomplish this with heterogeneous 3D integration. Our **third** insight is that the cost of such a solution does not need to be high. In our approach, we introduce a separate low-cost die that is dedicated to buffer placement, helping logic on the main die drive long interconnects without incurring buffer area and via blockage overheads. Our strategy places all logic cells on the *main (logic) die*, while buffers (particularly large buffers) are selectively placed on the *assisting (buffer) die*. As we demonstrate, this reduces both utilization of the main die and improves routability.

In support, the authors in [95] concluded that significant improvements in routability can help reduce chip area and thus manufacturing cost. Reduced routing congestion can also decrease crosstalk noise, reduce the critical area for shorts (thus improving yield) and decrease router runtime (thus improving overall design turnaround time). Figure 7.4 illustrates the structure of our proposed heterogeneous 3D integration. The two dies are manufactured at different technology nodes and integrated face-to-face. The *logic-die* is manufactured at advanced technology node, with high utilization and power density. It is placed close to the heat sink for better thermal conduction. The *buffer-die* has low utilization, with large buffers sparsely placed on it. The buffer die is then placed between the logic-die and the package, with extra silicon area consumed by TSVs.

Our key contributions include:

- Using a low-cost second die to house large buffers.
- Buffer-die planning and estimation for hetero-3D integration.
- Optimal buffer-die positioning and sizing.

- Dynamic selection of buffers to be placed on the buffer-die.
- Modifications to global routing to handle *elevated buffers*.
- Empirical evaluation based on ISPD 2011 benchmarks [108].
- Cost-benefits analysis of proposed techniques.

The remainder of this chapter is organized as follows. In Section 7.2, we briefly review relevant background information, formally present our problem, and introduce the overview of our 3D buffer-die approach. In Section 7.3, we present our optimization techniques for buffer-die sizing and buffer-die positioning. In Section 7.4, we outline our algorithm to selectively place buffers onto the buffer-die. In Section 7.5, we analyze the impact of multiple technology nodes on buffer insertion. In Section 7.6, we validate our proposed techniques by experiments on the ISPD 2011 gate-level netlists (benchmarks) and their variants. To simulate the buffer explosion problem, we describe our netlist rebuffering approach to account for practical timing correction. In Section 7.7, we discuss several observations regarding the potential technical obstacles when deploying the hetero-3D approach. In Section 7.8, we make our concluding remarks.

7.2 Overview

Given a gate-level design and a buffer-die area budget, we aim to reduce the routing of the original design. In this section, we outline the overview of our proposed solution, and then analyze issues regarding heterogeneous 3D integration.

Formally: Given (i) a logic-die L of fixed dimensions (X_L, Y_L), (ii) an area budget A_B^{max} for the buffer-die B, and (iii) a gate-level netlist with inserted buffers N placed onto L, we **determine** the location (x_B, y_B) and dimensions (X_B, Y_B) of B, and the subset of buffers from N to be placed onto B while **minimizing** the number of routing violations

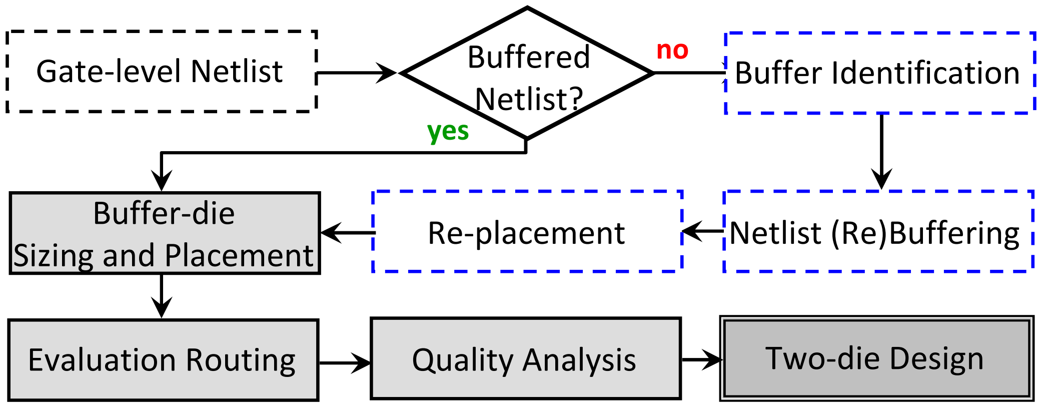


Figure 7.3: Work flow of our approach.

and routed wirelength on both L and B, **such that** $x_B \leq X_B$, $y_B \leq Y_B$, and $(X_B - x_B) \times (Y_B - y_B) \leq A_B^{max}$.

To do this, we rely on the heterogeneous face-to-face 3D integration of two dies with the buffer-die optimally sized and positioned relative to the logic-die. We use a lower-cost buffer-die to house buffers, which is smaller than and placed under the logic-die.

The optimization problem solved in our work is As illustrated in Figure 7.3, We start with the post-synthesized design, take the very limited number of buffer placeholders, and identify the set of relevant buffers. We then rebuffer those critical nets to be close to that of a modern IC design. Based on the netlist and buffer placement information, we optimize the location and shape of the buffer-die so as to maximize the number of buffers that can be moved to the buffer-die.

Netlist rebuffering. In Section 7.6, we explain how we identify buffer placeholders of the design and then for each buffer placeholder, insert additional buffers to account for timing. Based on our estimated inter-buffer distance l_{buf}, we clone buffers to rebuffer critical nets to match that of those in practical IC designs. Empirically, the percentage of buffers goes from 10% to over 30%.

Comparison	**Logic-die** L	**Buffer-die** B
technology node	new (32nm)	old (65nm)
unit cost	high	low
die size	large	small (60$\%$ of logic-die)
cells	logic & small buffers	mainly large buffers
metal layers	more (≥ 8)	fewer (≤ 4)
heat dissipation	large	small
position	top (near heat sink)	bottom (near package)

Table 7.1: Heterogeneity in 3D Integration.

Buffer-die sizing and placement. In Section 7.3, we describe optimal buffer-die positioning. For every considered buffer-die size, we find (i) the maximum number of contained buffers, and (ii) the location of B using a dynamic programming-based approach.

Buffer selection. In Section 7.4, we describe how we selectively move buffers from the logic-die to the buffer to the buffer-die to improve routability. Our buffer selection approach is based on the topology of each buffered net. Concurrently, we maintain the overall solution quality both on the logic-die and the buffer-die.

7.2.1 Heterogeneous 3D Integration

To integrate two dies of different technology nodes, we must first understand the requisite differences (summarized in Table 7.1). Compared to the cheaper buffer-die B, the more-expensive logic-die L (i) is of larger size, (ii) has more metal (routing) layers, (iii) achieves better performance, and (iv) dissipates more power. To mitigate the unnecessary (over)usage of L, we move buffers to B, since B dissipates less power. Here, L contains *all* the logic and most of the buffers, while B contains large and sparsely-placed buffers.

The two dies are face-to-face vertically integrated, as depicted in Figure 7.4. The two dies are connected by *Super-contacts* [104]; the package communicates with the chip by using TSVs on B. The substrate of L is adhered to the heat sink for thermal conduction, while B is placed below L and close to the package.

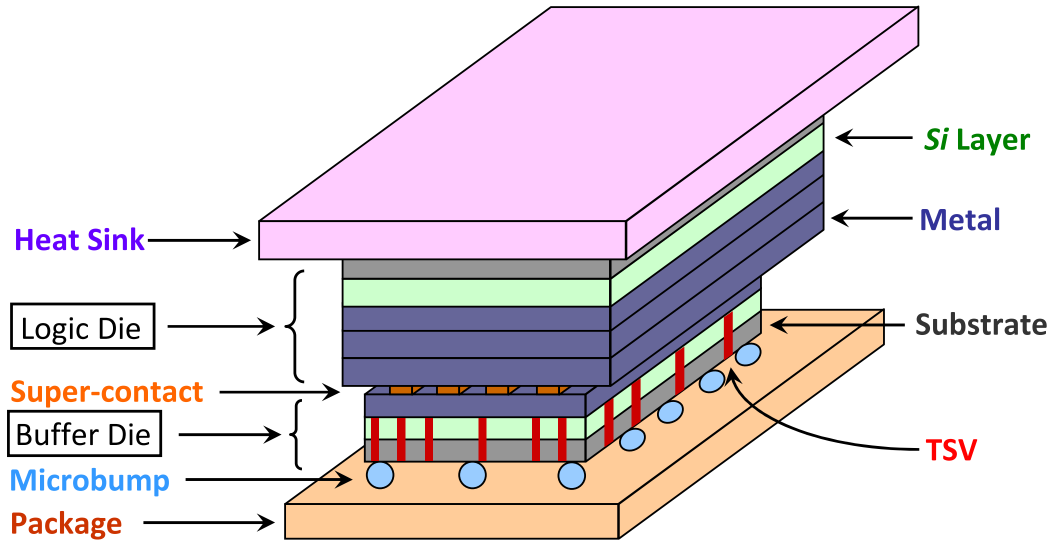


Figure 7.4: 3D face-to-face integration of logic and buffer-dies.

By moving buffers to the buffer-die, congestion on the logic die can be mitigated. For example, as illustrated in Figure 7.5(a), via stacks are necessary to connect high metal wires to a buffer on the silicon layer. However, if the buffer is moved to the buffer-die, we can remove one such via stack (Figure 7.5(b)). This reduces the total routed wirelength, improves routability, and reduces the routing turnaround time.

By introducing the buffer-die at an older technology node, buffers on the buffer-die will have larger gate and diffusion capacitance. However, the number of buffers on the buffer-die is small (10% according to our experiments), so the power overhead is limited. Therefore, we can use larger buffers on the buffer-die to provide enough driving strength, while the output slew rate can be estimated either by look-up table or some analytical modeling method [17]. Since the cost estimation is relatively difficult, we need to estimate the total cost of 3D integration based on technology node, dimensions and the number of metal layers of the buffer-die. In our work, we utilize the modeling method from [57] to calculate the manufacturing cost. In our buffer placement, we try to maximize the number of buffers being placed on the buffer-die. However, this maximization is constrained by

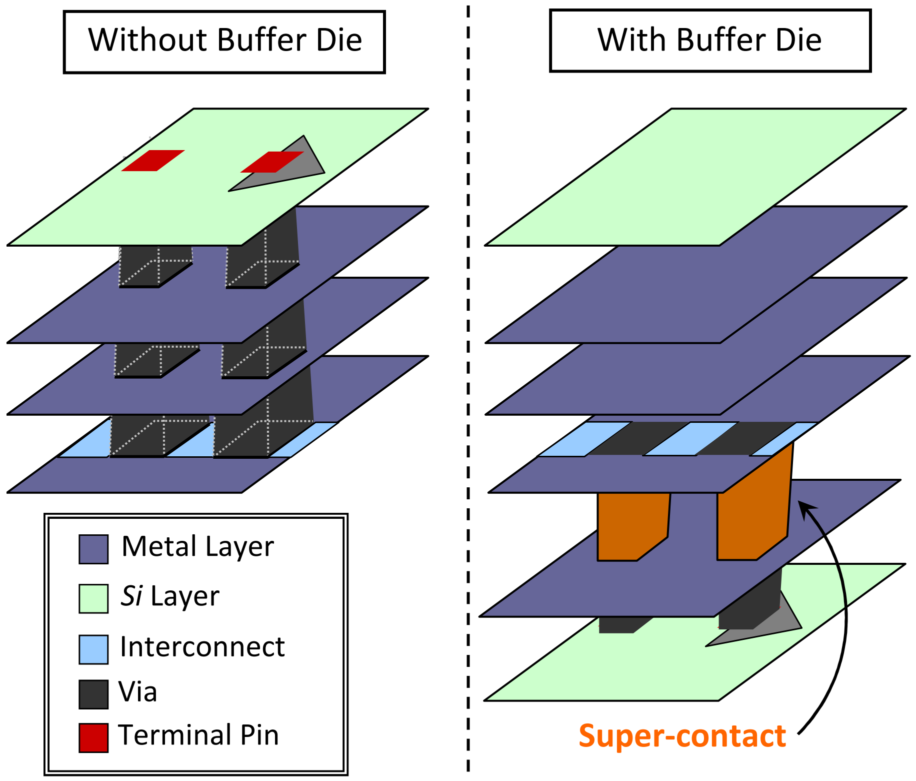


Figure 7.5: Interconnects on high metal layers are buffered (a) on the logic die with more vias consumed and (b) on the buffer-die through Super-contacts with less vias consumed.

buffer-die area and *chip-level buffer demand*, while the former comprises both buffer insertion area and TSV occupation area. Buffers need to be assigned to the logic-die, if the above constraints are violated.

7.3 Buffer-die Placement and Sizing

In this section, we discuss how to determine the position of the buffer-die – located directly below the logic-die – such that the buffer-die envelopes the largest number of buffers. For simplicity, we assume the buffer-die has equal width and height. This, however, can be easily extended to buffer dies with different aspect ratios.

7.3.1 Buffer-die Placement

To find the optimal position, we traverse all possible buffer-die locations and select the one containing most buffers. To count the number of buffers in an $m \times m$ region efficiently, we develop the dynamic programming-based method based on counting switching activity in [56]; the bottom-up procedure itself is given in Algorithm 8. We define (X_L, Y_L) to be the size of the logic-die L, and dim_B^{max} to be the maximum dimension of buffer-die B.

In the algorithm, the 3D matrix *buffer_count* with size (X_L, Y_L, dim_B^{max}) stores the total number of buffers, where each element *buffer_count*$[x][y][m]$ stores the number of buffers in an $m \times m$ region with lower-left corner (x, y). In the bottom-up approach, lines 1-5 finds the number of buffers located in each GCell (i.e., all 1×1 regions). Then, in lines 6-8, for each $m \times m$ region, we define the number of buffers in the region to be the summation of buffers in smaller regions. If m is even (lines 9-15), we write $m = 2k$, for some positive integer k, and the $m \times m$ region can be divided into four $(k \times k)$ regions, denoted as the lower-left (ll), lower-right (lr), top-left (tl), and top-right (tr); the total number of buffers in the region is the summation of the number of buffers in each quadrant. If m is odd (lines 16-23), we write $m = 2k + 1$, and the $m \times m$ region can be divided into non-equal, overlapping regions; lines 22-23 removes the overlapping portion and sums together the subregions. The summation and inclusion/exclusion is demonstrated in Figure 7.6.

7.3.2 Buffer-die Sizing

We study the impact of sizing the buffer-die using the placement solutions of four ISPD 2011 testcases, SUPERBLUE{ 1,2,4,5}, all of which are rebuffered by our approach in Section 7.6 to increase the buffer number. Under different sizes, the buffer-die is optimally placed inside the logic-die. We illustrate our statistics of impacts of buffer-die sizing in Figure 7.7, where the ratio of buffer-die dimension over logic-die dimension is linearly

Algorithm 8 Counts the number of buffers in an $m \times m$ region.

Input: Maximum dimensions (X_L, Y_L) of logic-die L, maximum dimension dim_B^{max} of buffer-die B

Output: Matrix *buffer_count* of size (X_L, Y_L, dim_B^{max}), where element (x, y, m) stores the number of buffers in the $m \times m$ region with lower-left corner (x, y).

```
for $x$ from $1 \rightarrow X_L$ do
    for $y$ from $1 \rightarrow Y_L$ do
        $buffer\_count[x][y][1] = \texttt{NUM\_BUFFERS\_IN\_GCELL}(x,y)$;
    end for
end for
for $m$ from $2 \rightarrow dim_B^{max}$ do
    for $x$ from $1 \rightarrow X_L - m + 1$ do
        for $y$ from $1 \rightarrow Y_L - m + 1$ do
            if $m \mod 2 == 0$ then
                $m = 2k$;
                $ll = buffer\_count[x][y][k]$;
                $lr = buffer\_count[x + k][y][k]$;
                $tl = buffer\_count[x][y + k][k]$;
                $tr = buffer\_count[x + k][y + k][k]$;
                $buffer\_count[x][y][m] = ll + lr + tl + tr$;
            else
                $m = 2k + 1$;
                $ll = buffer\_count[x][y][k]$;
                $lr = buffer\_count[x + k][y][k + 1]$;
                $tl = buffer\_count[x][y + k][k + 1]$;
                $tr = buffer\_count[x + k + 1][y + k + 1][k]$;
                $cn = buffer\_count[x + k][y + k][1]$;
                $buffer\_count[x][y][m] = ll + lr + tl + tr - cn$;
            end if
        end for
    end for
end for
```

increased. By adding a maximally sized buffer-die, we can on average remove 90% of buffers from the logic-die, as shown in Figure 7.7(a). The buffers are sparsely placed on the buffer-die and the buffer-die utilization is very small (on average, 7% when the buffer-die dimension is 70% of the logic-die dimension, as illustrated in Figure 7.7(b)). However, the actual buffer-die utilization should be larger, because buffers on the old die would be

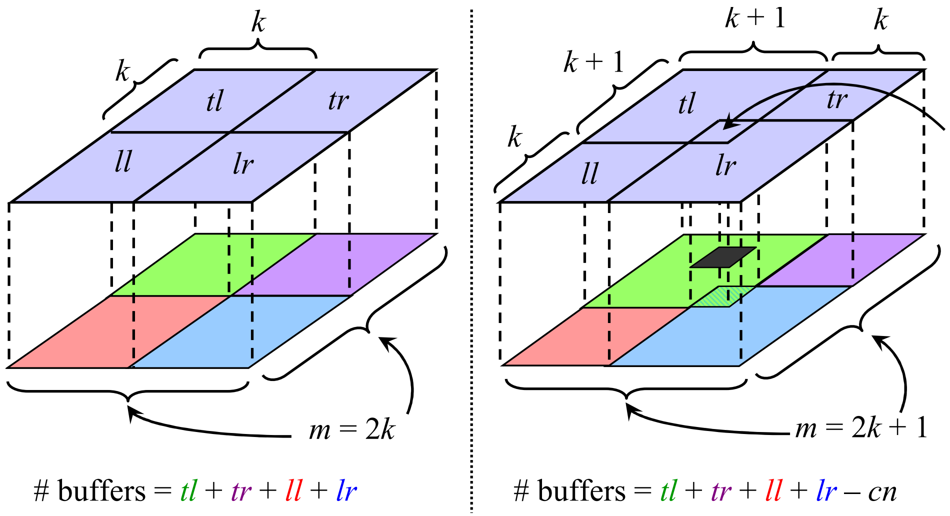

k
k
tl
tr
ll
lr
m = 2k
# buffers = tl + tr + ll + lr
k + 1
k + 1
k
k
tl
tr
cn
ll
lr
m = 2k + 1
# buffers = tl + tr + ll + lr – cn

Figure 7.6: Illustration of counting buffers in an $m \times m$ region. The left side shows when $m = 2k$ is even – the number of buffers in the region is the sum of 4 disjoint $k \times k$ quadrants. The right side shows when $m = 2k + 1$ is odd – the number of buffers in the region is the sum of 4 subregions, two of which are non-disjoint. The duplication is removed by subtracting the number of buffers in the overlapping (center) region.

manufactured at an older technology node. If we assume one technology node difference between the logic- and buffer-dies, the unit buffer area on the buffer-die will be $2\times$ and the buffer-die utilization will increase from 7% to 14%. Addressing the gap of output signal slew between technology nodes will further enlarge the buffers on the buffer-die.

Figure 7.8 shows the location of buffer-die on the GCell-wise buffer distribution map of the logic-die. In the figure, a **white** dot denotes that there is no buffer in the corresponding GCell, a **red** dot denotes that there is one buffer in the GCell while a **black** dot denotes that more than one buffer are located in the GCell. Each **cyan** contour denotes the optimal placement of the buffer-die under different die sizes. We can also observe from the figure that buffers are sparsely placed on the buffer-die. As the size of the buffer-die increases, the number of buffers enclosed will increase, and the utilization will decrease.

7.4 Buffer Selection

Buffers inside the buffer-die contour will be selectively placed from the logic-die to the buffer-die, in order to minimize routing violations and total wirelength. Our buffer selection algorithm ranks all the buffers in the design, and then chooses the best candidates based on only the routing capacity of the buffer-die; we observe that the available area on the buffer-die is not a constraint, despite larger buffer sizes in the older technology node. Buffers are sparsely placed on the buffer-die with very low die utilization. As a result, the amount of whitespace in between is large, which can satisfy area requirements when upsizing buffers and reduce the likelihood of any overlap between buffers.

To determine which buffers should move to the buffer-die, we first determine their GCell locations on the global routing grid. Second, we partition the set of buffered nets into three sets: (i) **Single-GCell net** (SG) of which pins are contained within a single GCell on a single metal layer, (ii) **Single-buffer net** (SB) of which only one buffer is contained, and (iii) **Multi-buffer net** (MB) of which more than one buffers are used to buffer the interconnect. Among the three groups, we give the highest priority to buffers which belongs to multi-buffer net (group (iii)) and the lowest priority to those in group

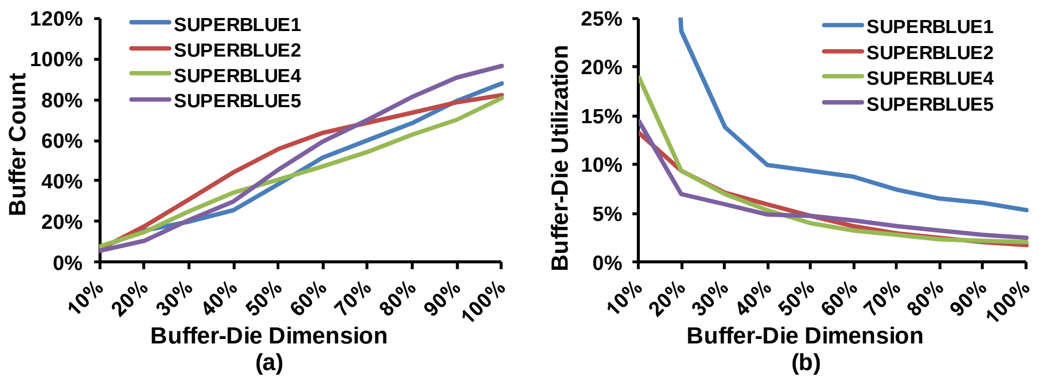


Figure 7.7: Statistics of the optimally placed buffer-die under different dimensions: (a) % of buffers in the buffer-die (b) utilization of the buffer-die.

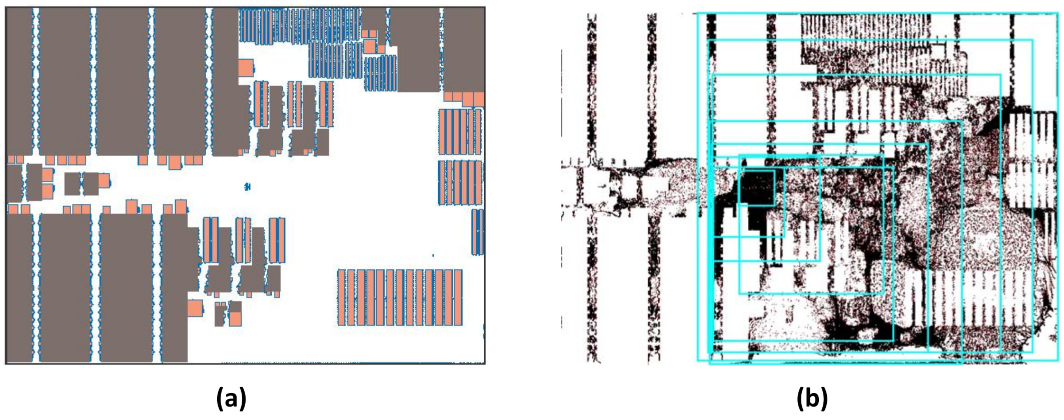


Figure 7.8: Comparison of (a) floorplan and (b) buffer distribution map of SUPERBLUE1.

(i). That is, we consider buffers in (iii) before considering those in (ii). Third, within each group, we assign each buffer a local priority based on the amount of potential improvement it has. If the benefit is positive, the buffer is moved.

For buffers in (iii), the local priority is based on the number of buffers within each net and the bounding box of the net. Ideally, every involved buffer is elevated, as the cost of traversing to the buffer-die is amortized. However, if constrained, we choose the buffers that are contained in nets that use the largest number of buffers. To break ties, we select the net with the largest bounding box. For buffers in (ii), the local priority is based on local congestion and potential via reduction. Specifically, we compute the ratio between the total number of segments routed in the top half and the bottom half of the design; the priority is the net difference between the two. Here, we observe that the cost of traversing to the buffer-die is more pronounced, as only one buffer is involved. However, if the net was already routed in the top metal layers, then the cost is mitigated. Moreover, to improve routability, the net can be coerced to a higher metal layer if there is congestion on the lower metal layer. To quantify this potential, we use a previous routed solution and analyze the congestion at each routed segment. We give the buffer higher priority if its connecting nets are either routed in the top metal layers or if the top metal layers have less congestion. For buffers in (i), we do not assign a local priority, as we do not move them. Intuitively, if these buffers were elevated, they would consume routing resources not previously needed.

7.5 Buffer Transformation

In our hetero-3D approach, we assume that there is difference of one technology node between the two dies. Buffers on the buffer-die will be synthesized and manufactured at an old technology node, which requires larger device size due to constraint on feature and signal output slew. Meanwhile, the driving length of buffers is increased, because of reduced interconnect resistivity and increased gate capacitance. Therefore the actual number of buffers placed on the buffer-die is likely to be smaller than that of those originally assigned to the buffer-die. Figure 7.9 illustrates how the buffer chain changes by placing onto the buffer-die.

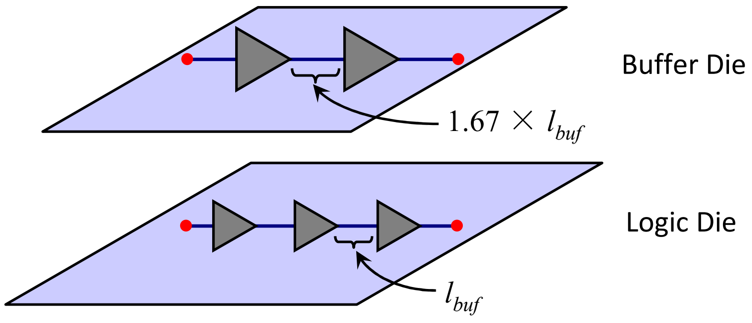


Figure 7.9: Technology adjustment of buffer chains.

7.5.1 Inter-Buffer Distance Estimation

We now discuss the scaling of inter-buffer distance between two successive technology nodes. Let T be the Elmore delay of a long buffered interconnect which consists of a couple of identical stages. Each stage includes a buffer driving a wire segment, which is of length l_{buf}. Assume that the effective resistance and gate capacitance of the buffer are R_g and C_g, while the resistance and capacitance per unit wirelength are R_w and C_w, respectively. Setting $\frac{dT}{dl} = 0$ yields the optimal inter-buffer distance of $l_{buf} = \sqrt{\frac{R_g \times C_g}{R_w \times C_w}}$ [6]. In ideal technology scaling, we have R_g and C_g scaled by $1.0\times$ and $0.7\times$ per node, while R_w and C_w respectively scale by $2.0\times$ and $1.0\times$ per node. As a result, we have l_{buf} ideally scaled down by $1.67\times$ per technology node.[3]

7.5.2 Buffer Upsizing

Here we analyze how much a buffer will be upsized while moving it from the logic-die to the buffer-die. The size of a buffer is dependent on the technology node and the required signal slew that it should provide. Logic cells designed at a new technology node will require an input signal with smaller transition time, which is essentially linear with MOSFET intrinsic delay (CV/I). From ITRS 2010 PIDS tables [53], we see that CV/I is reduced by 13% per year, or $1.44\times$ per technology node. As a result, we assume that the output slew rate of buffers on the buffer-die is reduced by $1.44\times$. From output-slew tables in $TSMC$ $65nm$ liberty files, we obtain that a basic buffer $BUFFDx$ must be upsized by $1.28\times$ to supply signal with slew rate reduced by $1.44\times$. Furthermore, $1.4\times$ scaling of feature dimension implies $2\times$ scaling of buffer area; we therefore end up with buffers on the buffer-die upsized by $1.4 \times 1.4 \times 1.28 = 2.51\times$. In our testbed, we increase the placement row height on the buffer-die by $1.4\times$, while the width of buffers on the buffer-die is scaled up by $1.79\times$.

7.6 Empirical Validation

To evaluate the effectiveness of the buffer-die, we modified the ISPD 2011 Benchmarks [108] in accordance with our proposed methodology, and routed the modified benchmarks. Table 7.2 summarizes our empirical observations.

[3]We have verified our estimation of the scaling of inter-buffer distance between 22nm and 16nm based on ITRS 2010 PIDS2 and INTC6 tables. The resultant scaling factor is $1.47\times$, with the discrepancy mainly caused by our assumption of ideal scaling in the estimation.

		Original			Modified			w/o Buffer Die		w/ Buffer Die	
BENCHMARK	# Non-Buffers	Total # Buffers	%age Buffers	%age Area	Total # Buffers	%age Buffers	%age Area	Total OF	RtWL (e7)	Total OF	RtWL (e7)
SUPERBLUE1	767798	79643	9.40	0.99	490668	38.99	5.51	0	1.54	0	1.54
SUPERBLUE2	918443	95586	9.43	0.50	388932	29.75	1.99	6022678	2.72	5943724	2.72
SUPERBLUE4	561189	39031	6.50	0.70	166490	22.88	2.71	6652	1.09	2182	1.11
SUPERBLUE5	684609	87848	11.37	0.82	344299	33.47	3.37	961342	1.66	806112	1.74
SUPERBLUE10	1050027	79117	7.01	0.53	479864	31.37	3.17	1142604	2.71	1056706	2.72
SUPERBLUE12	1196518	96915	7.49	1.62	1190255	49.87	16.47	1498992	2.44	1401058	2.45
SUPERBLUE15	1075642	48321	4.30	1.17	175290	14.01	5.14	0	1.73	0	1.73
SUPERBLUE18	455586	27866	5.76	0.62	187133	29.12	4.13	316	1.23	0	1.23
Average						31.18	5.31	$1.20\times$	$0.99\times$	$1.00\times$	$1.00\times$

Table 7.2: Empirical results of our buffer insertion and routability experiments. Here, RtWL is the summation of routed horizontal and vertical segments, and the number of vias. We ran every benchmark with a hard limit of 60 minutes.

Netlist rebuffering. Our work seeks to compensate the negative impact caused by buffer explosion. However, the ISPD 2011 gate-level netlists are at post-synthesis design stage, and have limited placeholder buffers in the netlist. Therefore, we must first identify placeholder buffers, and then use a buffer cloning technique to rebuffer all the nets. We define the length of each inter-buffer wire segment to be l_{buf}. The estimation of l_{buf}, however, is difficult, as we do not have information at which technology node these netlists have been synthesized at. As a result, we empirically define l_{buf} and uniformly insert buffers for every distance of l_{buf}.

Placeholder buffer identification. We require buffer information from the netlist as input to the buffer-die sizing optimization. However, there is only area and pin count for each instance, while library information is hidden. Moreover, instance pin count is not accurate due to excluded networks in the benchmark (e.g., clock network, scan chain).

Given that the row height is nine units, the size of a NAND gate is estimated as roughly 40 units of area. The size of a flip-flop is typically $5\times$ that of a NAND gate (four unit gates and one scan-chain MUX), which is about 200 units of area. Meanwhile, we assume that typically between 15% and 20% of design instances are flip-flops.

The statistics of instance counts for different cell areas and pin counts are shown in Figure 7.10, for the SUPERBLUE1 benchmark.[4] We can see a peak at 220 units of area

[4]The number of instances with pin count = 1 is very small and we do not show it in the figure.

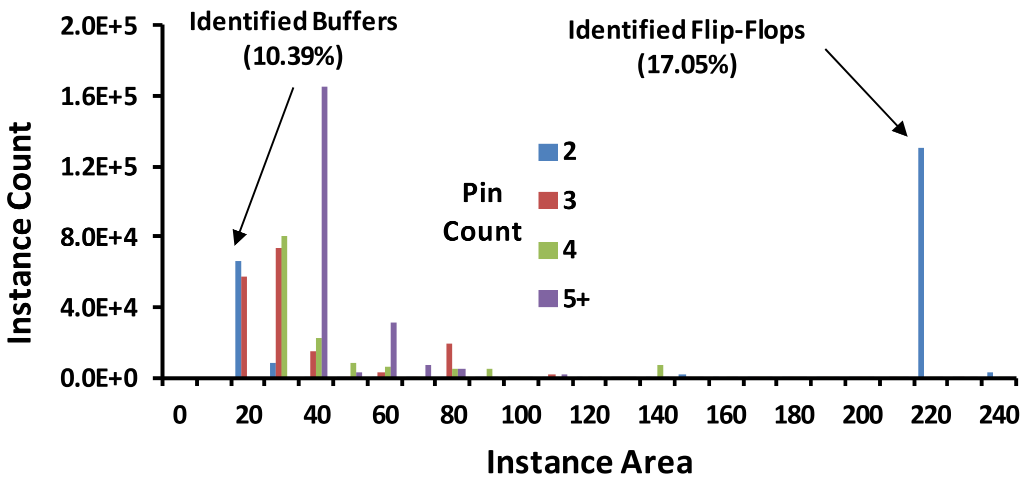


Figure 7.10: Cell size and pin count distribution in SUPERBLUE1.

($5.5\times$ the NAND gate size), for 17.05% of the instances in the testcase. Both instance number and area ratio support our assumption, and we identify these large-size, two-pin instances as flip-flops. The remaining two-pin instances are of smaller sizes and are identified as buffers. Other instances (three-pin or more) are assumed to be combinational cells. The ISPD 2011 testcases are generated at an early stage of the design flow (post-synthesis) without any timing correction [3], and only a few buffers (about 10%) are found. We also run tests on other benchmarks and obtained very similar statistics, which further verifies our assumptions. Finally, we have obtained flip-flop information from IBM [108] to verify our identification; our false positive rate is 0.3%, and our false negative rate is less than 1.5%. This supports a conclusion of high accuracy in our identification of flip-flops and buffers in the benchmark netlists.

Timing correction. Given a set of buffer placeholders, we remove each buffer placeholder-chain, and insert a buffer-chain based on the critical length. Here, we define a existing buffer placeholder-chain as the longest segment of buffer placeholders. We define l_{buf} as the average length of non-buffered nets. To better model the buffer explosion and consider more practical cases, we perform rebuffering critical interconnects to increase the area used by buffers. In practice, the placeholder buffers usually account for 10% of the netlist; after our buffer insertion procedure, the percentage increases to over 30%. We estimate accuracy of our methodology based on the buffer information we obtained from [3]. It shows that the percentage of buffers inserted by our rebuffering approach matches that of modern IC design cases.

Experimental setup. Our single-threaded buffer insertion and buffer placement tools are implemented in C and C++ and compiled with g++ 4.3.2. After buffer insertion, we use SimPLR [65] to generate a legal and routability-aware placement. To evaluate the modified benchmarks, we adapt BFG-R [43] handle (i) the 2011 benchmark constraints, including virtual (elevated) pins and layer-specific widths and spacings, and (ii) create a routing model of the buffer-die. To represent the buffer-die, we create an additional tier of metal layers, located directly below the topmost metal layer (of the logic-die). Here, we define a tier as a pair of horizontal and vertical layers, with each layer having a preferred routing direction. Since this is an older technology node, we use a comparable wire width and wire spacing to those of lower metal layers.

Our **buffer insertion** results are in the *Original* and *Modified* columns of Table 7.2. Before buffer insertion, buffers compromised of less than 10% of the original benchmarks. To rebuffer the nets, we followed the approach outlined in Section 7.6, and set the inter-buffer distance to be the average length of all non-critical nets. After buffer insertion, buffers make up about 30% of the design's cells. Our **routability** results are in the *w/o Buffer Die* and *w/ Buffer Die* columns. Here, *Total OF* is the total number of additional required routing tracks, and *RtWL* is sum of routing tracks and vias. Since the buffer die uses an older technology generation, we set to two times the capacity of the topmost routing layer, and the routing pitch that of the middle routing layers. On benchmarks that originally have overflow, our approach can reduce the total overflow of the designs by

20%. Our approach is also able to route SUPERBLUE18 without any violations. In all cases, we maintain similar same solution quality as before.

When choosing **buffer-die** sizes, we base our decision on whether the design is originally routable. If not routable, we choose a larger-sized die (comparable to the logic die, $> 2/3$ the dimension) to better alleviate routing congestion and mitigate overflow. However, if the original design is routable, then only a small ($< 1/2$ the dimension) buffer-die is needed to either maintain or improve routability.

7.7 Open Technical Issues Associated with the Hetero-3D Approach

While our approach improves 2D congestion, total wirelength and routing turnaround time, it also opens up several technical issues which any production methodology would need to work through. For instance, the dimensions of TSVs and Super-contacts are $10\times$ to $100\times$ larger than those of on-chip metal wires or devices; given the congestion on the topmost (thick) tier of the logic-die due to power/ground distribution, our stacked-die configuration may be prone to vertical congestion. Power signal delivery, die-to-die data signal transmission as well as signal IO all consume both TSVs and Super-contacts. When instantiated at the older technology node, each instance on the buffer-die will consume more power than it would have on the logic-die, which causes a power overhead to our approach. As has been well-studied in recent years, increased power density can cause higher peak temperatures and lower system reliability. In this supplementary section, we acknowledge some of the open technical issues that remain to be addressed.

7.7.1 3D Congestion Estimation

As shown in Figure 7.4, power is delivered from package to the buffer-die by TSVs and from the buffer-die to the logic-die by Super-contacts, and are used for the following.
Power delivery. TSVs deliver power from the package to the buffer-die, Super-contacts

deliver power from the buffer-die to the logic-die. From [45] we assume that TSV connections are uniformly distributed with a density of 80 per square millimeter. The length, diameter, and dielectric liner thickness of a typical TSV connection are assumed to be $50\mu m$, $5\mu m$, and $0.12\mu m$, respectively. A TSV is modeled as an RLC element. The resistance and inductance are in series while capacitance is connected to the substrate (global ground). From the analytical TSV model in [64], the values of RLC are calculated to be $47m\Omega$, $34pH$, and $88fF$, respectively. Here, we assume that the worst case voltage drop on the two dies are within 5% of the voltage supply. From the sum of current demands of the two dies and the resistance of one TSV, we can estimate how many TSVs are needed for power delivery to the buffer-die and logic-die. For the Tezzaron Super-Contact interconnect technology [104], pitch is $3\mu m$, capacitance is $2 - 3fF$ and resistance is 0.3Ω. We may estimate the IR-drop caused by Super-contacts in the same way as with TSVs. When mapped to 65LP technology, we estimate that the SUPERBLUE2 benchmark (approximately 3.8 $\times$ 5.4 mm^2 in 45nm technology) has a current demand of roughly $1.5A$. As a result, only a fair number of power TSVs ($<$ 20) and power Super-contacts ($<$ 200) are needed, even with very stringent IR-drop constraints.

Buffer connection. Super-contacts transmit data signal between logic cells on the logic-die and buffers on the buffer-die. As also shown in Figure 7.9, placing each buffer chain on the buffer-die will consume two Super-contacts. We try to minimize the number of vertical connections by putting long buffer chains onto the buffer-die while keeping short buffer chains on the logic-die. Currently, we do not verify the legality of buffer placement with respect to overlaps on the buffer-die (i.e., we only elevate the buffer locations and/or apply the critical repeater length-based buffer insertion on the buffer-die). However, we analyze the vertical routing resources (number of available Super-contacts) and require that they satisfy the demand from die-to-die connections.

7.7.2 Power and Thermal Estimation

The hetero-3D approach can potentially incurs power overheads, since buffers at a coarser technology node are more power consuming. However, the number of buffers on the buffer-die is quite small, which limits the power overhead. Based on our two-die placement and routing solution, we obtain the number of instances and total wirelength of interconnects on the two dies. We use the power model from [55] to estimate the static and dynamic power consumed by devices and interconnects on the two dies. The experimental results in Section 7.6 show that power overhead of our approach is small.

We adapt the power model of [55] to estimate power consumption of the two dies, with the following assumptions obtained from it. We estimate power from the total number of transistors and total wirelength in each design after global routing. A typical 2-input NAND gate is of dimension 4×9 and comprises four transistors. We add up the total area of all movable instances after rebuffering the netlist, and estimate the total number of logic transistors. The total memory area is calculated as the sum of areas of all the fixed blocks. Area overhead (for peripheral logic, etc.) in a memory block is assumed to be $1.6\times$, while each bitcell comprises six transistors with a total area of $\frac{1}{3}$ of that of a NAND gate. As a result, we obtain the total number of transistors (logic plus memory) by adding the above two numbers. As in [55], a typical NAND gate occupies a rectangular region of $9F \times 20F$, where F is the Metal-1 half-pitch ($m1hp$). In 45nm technology, $m1hp$ is 70nm, and thus the height of a NAND gate is $1.4\mu m$. In the ISPD 2011 benchmark suite the height of a placement row is fixed to be 9 units, therefore one unit length in the benchmark equals $1.4\mu m/9 = 155.56nm$ under 45nm technology. Similarly, we can calculate one unit length under 65nm technology as $160nm$. Since the dimension of a GCell is 32 units, i.e., $4977nm$, we can calculate the total global routing wirelength in terms of GCell dimensions. We use SUPERBLUE2 as our testcase using 45nm technology on the logic die. Its power dissipation is estimated at 1.75 watts. The power dissipation of the buffer die stacked on top of SUPERBLUE2 is estimated at 0.016 watts.

Finally, we may analyze the thermal conditions of our hetero-3D integration by using HotSpot 5.0 [46] at the block-level. We assume that there is only a single core for each die, with the power dissipation uniformly distributed over each die. Based on the power numbers above for the two dies, we are able to simulate the thermal conditions of our hetero-3D integration and calculate the peak temperature (42C). The results show that our 3D IC approach will not cause any additional thermal problems.

7.8 Conclusions

Our research studies the interaction of 3D integration with key interconnect trends in modern and future ICs. Where previous research on interconnect stacking employs rather coarse interconnect model, we develop a detailed testbench that replicates the buffer explosion problem and ensuing routing congestion. In particular, we use a leading-edge global router and the ISPD 2011 benchmark suite from IBM to quantify the impact on routing congestion of via stacks required to buffer high-metal interconnect. We propose a novel technique by which 3D die stacking can alleviate these routing obstructions. This technique provisions for a small second die fabricated at a coarser technology node and consisting entirely of buffers, which will be connected to high-metal interconnect through Super-contacts. These techniques are supported by a powerful optimization backplane, which decreases their overhead.

Moreover, we contribute models of buffer identification, netlist rebuffering, buffer-die placement and buffer selection. These models are used to provide a detailed cost-benefit analysis of our proposal for heterogeneous 3D integration.

CHAPTER VIII

Conclusions and Future Research Directions

Within the VLSI design flow, global routing is a critical step that heavily impacts the final quality of the chip. Given a placed solution, the global routing stage determines the routability of the design, and optimizes the design for metrics like power and minimal resource usage. Within the context of chip design, global routing is applicable to many other physical design-steps, such as estimating routability within global placement. This dissertation (i) introduced the fundamentals of global routing and its relevant literature, (ii) discussed enhancements to global routing, and (iii) explored global routing extensions to different design and optimization flows.

8.1 Summary of Our Contributions

Standalone global routing. We presented a number of different optimizations, techniques, and algorithms to significantly improve solution quality and scalability within different routing approaches. In Chapter III, we used integer linear programming to optimally select the best route from a set of candidates for each net. Our scalable implementation Sidewinder explicitly improved the number of vias with moderate runtime and resource overhead. In Chapter IV, we developed an edge-centric Lagrangian-relaxation routing formulation that was able to handle millions of nets, encapsulate modern technology constraints, and efficiently control quality. Our implementation BFG-R effectively reduced routing congestion while maintaining a low wirelength and runtime profile. BFG-R was also one of two evaluation routers in the DAC 2012 Routability-driven Contest [109].

Placement-and-routing integration. Routability can be further improved at design-flow stages, such as placement. However, the router must be not only accurate, but also fast, as it will be invoked several times. In Chapter V, we presented our complete routability-driven placement flow, where we integrated a lighter version of BFG-R (Chapter IV) with a state-of-the-art global placer [66] and detailed placer [89] to significantly improve congestion. Our lightweight implementation SimPLR uses the router to proactively avoid causing difficult-to-route regions by bloating cells and controlling cell movement. In Chapter VI, we improved the speed of constructive routing estimation by an order of magnitude through linear-time cache-friendly algorithms and developed several techniques to relieve different types of congestion. Our coordinated place-and-route framework combines standalone components to systematically reduce the complexity of placement and routing.

Extensions to 3D technology. We addressed the *buffer explosion* problem by integrating multiple technology nodes. In Chapter VII, we presented out hetero-3D approach to selectively moving buffers to an older-technology die to reduce routing congestion and costs. By adapting the global router to account for new technology constraints, we were able to accurately estimate the routing feasibility of our idea, and can reduce the total chip area.

8.2 Directions for Future Work

We envision several different directions for future research.

In **parallelizing** global routing, one option is to divide the solution space, and to evaluate each partition in parallel. Using multiple threads, there are two general approaches. The first is using a coarse-grained partition, where each thread considers a portion of the layout space; the second is using a fine-grained partition, where each thread considers a net. In previous literature, PGRIP [119] followed the former approach, whereas PGR [77] followed the latter approach. Based on empirical results and ease of implementation, the former approach is easier to implement, whereas the latter can require more resources and threads. However, the latter approach can lead to better runtime improvements. To take advantage of both approaches, we consider a hybrid technique. First, to maintain the spirit of practicality, we tailor the number of threads to the experimental setting (e.g., workstation). Second, each thread can be assigned to a layout region or cluster, depending on the situation or progress of the router. This allows us to customize our efforts to the areas that require the greatest optimization effort.

Using **graphics processing units (GPUs)** can significantly decrease router runtime [37]. Here, the authors assign a net to a single thread in the GPU, and route sets of nets such that the bounding box of each net has no overlap with the bounding box of any other nets within the set. Therefore, the authors bypass the need to handle collisions. To expand on this framework, we propose to have the GPU perform more tasks, and on a finer granularity. First, we notice that we can quickly update history costs and other relevant costs (e.g., net costs) in parallel. Second, we can adopt this method by assigning each two- and three-pin net and subnet to a thread. That is, if a net is a two- or three-pin net, then it is assigned to one thread. Otherwise, it is decomposed into two- or three-pin subnets, and each of its subnets are assigned to a thread.

Handling **timing constraints** or limiting nets that have a large number of detours is an important consideration. In general, global routers are blind to this aspect, as their primary objective is routability, followed by the minimization of total wirelength. Therefore, nets are purposely detoured without discrimination. However, this can hurt performance, since some are considered *timing-critical*, and cannot exceed some given length.

To account for timing-related constraints, one approach within global routing is to impose a bounding box or length constraint for specific nets. However, this can harm overall routability, as these nets can use resources that are requisite for other nets. To address this, we propose to rip-up and reroute these nets more often, and impose the strict length constraint. With the increasing history costs within a local region, other nets should eventually be pushed away, encouraging them to detour. Another type of timing constraint is to impose nets to specific layers. A previous work GLADE [15] has accounted for net-layer restrictions. To extend this idea, we propose to modify layer assignment such that it handles being restricted to different layers. For example, if a net is restricted to being specific layers, then the edges on those layers have finite cost.

Routability-driven placement methods still have areas of improvement. One direction would be to embed congestion information into the placement objective function. However, this would require one (or multiple) router invocation at every iteration, thereby requiring congestion estimation be fast. Our techniques can be extended to further improve runtime by testing incremental routing *without* preserving the net order, and incorporating history-cost updates.

BIBLIOGRAPHY

BIBLIOGRAPHY

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

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

[3] C. J. Alpert, IBM, *personal communication*, November 2011.

[4] C. J. Alpert, Z. Li, M. D. Moffitt, G.-J. Nam, J. A. Roy and G. Tellez, "What Makes a Design Difficult to Route", *ISPD* 2010, pp. 7-12.

[5] C. J. Alpert, D. P. Mehta and S. S. Sapatnekar (eds.), *Handbook of Algorithms for VLSI Physical Design Automation*, CRC Press, 2008.

[6] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*, Addison-Wesley, 1990.

[7] J. Bentley, "Programming Pearls: Algorithm Design Techniques", *ACM* 27(9) (1984), pp. 865-873.

[8] S. Bobba, A. Chakraborty, O. Thomas, P. Batude, T. Ernst, O. Faynot, D. Z. Pan and G. D. Micheli, "CELONCEL: Effective Design Technique for 3D Monolithic Integration Targeting High Performance Integrated Circuits", *ASP-DAC* 2011, pp. 336-343.

[9] U. Brenner and A. Rohe, "An Effective Congestion Driven Placement Framework", *ISPD* 2002, pp. 6-11.

[10] M. Burstein and R. Pelavin, "Hierarchical Wire Routing", *TCAD* 2(4) (1983), pp. 223-234.

[11] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wire-length Estimations for Row-based Placement", *TCAD* 18(9) (1999), pp. 1265-1278.

[12] coalesCgrip: A Tool for Routing Congestion Analysis. `homepages.cae.wisc.edu/~adavoodi/gr/cgrip.htm`

[13] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze and M. Xie, "mPL6: Enhanced Multi-level Mixed-size Placement with Congestion Control", *Modern Circuit Placement* 4 (2007), pp. 247-288.

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

[15] Y.-J. Chang, T.-H. Lee and T.-C. Wang, "GLADE: A Modern Global Router Considering Layer Objectives", *ICCAD* 2010, pp. 319-323.

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

[17] M. Chen, Y. Yi, W. Zhao and D. Ma, "Variation-Aware Deep Nanometer Gate Performq ance Modeling: An Analytical Approach", *VLSI-DAT* 2011, pp. 1-4.

[18] C. E. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling", *ICCAD* 1994, pp. 650-695.

[19] L. Cheng, L. Deng and M. D. F. Wong, "Floorplanning for 3-D VLSI Design", *ASPDAC* 2005, pp. 405-411.

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

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

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

[23] P. Christie and D. Stroobandt, "The Interpretation and Application of Rent's Rule", *TVLSI* 8(6) (2000), pp. 639-648.

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

[25] C. C. N. Chu and Y.-C. Wong, "Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design", *ISPD* 2005, pp. 28-35.

[26] Y.-L. Chuang, G.-J. Nam, C. J. Alpert, Y.-W. Chang, J. A. Roy and N. Viswanathan, "Design-hierarchy Aware Mixed-size Placement for Routability Optimization", *ICCAD* 2010, pp. 663-668.

[27] J. Cong and G. Luo, "A Multilevel Analytical Placement for 3D ICs", *ASP-DAC* 2009, pp. 361-366.

[28] J. Cong, J. Wei and Y. Zhang, "A Thermal-driven Floorplanning Algorithm for 3D ICs", *ICCAD* 2004, pp. 306-313.

[29] J. Cong and Y. Zhang, "Thermal-driven Multilevel Routing for 3-D ICs", *ASP-DAC* 2005, pp. 121-126.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, Second Edition, MIT Press and McGraw-Hill, 2001.

[31] ILOG CPLEX: High-performance Software for Mathematical Programming and Optimization. `http://www.ilog.com/products/cplex`

[32] L. Dagum and R. Menon, "OpenMP: An Industry Standard API for Shared-memory Programming," *Computational Science and Engineering* 1998, pp. 46-55.

[33] K.-R. Dai, C.-H. Lu and Y.-L. Li, "GRPlacer: Improving Routability and Wirelength of Global Routing with Circuit Replacement", *ICCAD* 2009, pp. 351-356.

[34] J. A. Davis, V. K. De and J. D. Meindl, "A Stochastic Wire-length Distribution for Gigascale Integration (GSI). I. Derivation and Validation", *Electronic Devices* 45(3) (1998), pp. 580-589.

[35] D. E. Donath, "Placement and Average Interconnection Lengths of Computer Logic", *Circuits and Systems* 26 (1979), pp. 271-277.

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

[37] Y. D. Han, D. M. Ancajas, K. Chakraborty and S. Roy, "Exploring High Throughput Computing Paradigm for Global Routing", *ICCAD* 2011, pp. 298-305.

[38] X. He, T. Huang, L. Xiao, H. Tian, G. Cui and E. F. Young, "Ripple: An Effective Routability-driven Placer by Iterative Cell Movement", *ICCAD* 2011, pp. 74-79.

[39] W. Hou, H. Yu, X. Hong, Y. Cai, W. Wu, J. Gu and W. H. Kao, "A New Congestion-driven Placement Algorithm Based on Cell Inflation", *ASP-DAC* 2001, pp. 723-728.

[40] B. Hu and M. Marck-Sadowska, "Congestion Minimization During Placement Without Estimation", *ICCAD* 2002, pp. 739-745.

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

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

[43] J. Hu, J. A. Roy and I. L. Markov, "Completing High-quality Global Routes", *ISPD* 2010, pp. 35-41.

[44] Y. C. Hu, Y. L. Chung and M. C. Chi, "A Multilevel Multiplayer Partitioning Algorithm for Three Dimensional Integrated Circuits", *ISQED* 2010, pp. 483-487.

[45] X. Hu, P. Du and C.-K. Cheng, "Exploring the Rogue Wave Phenomenon in 3D Power Distribution Networks", *EPEPS* 2010, pp. 57-60.

[46] W. Huang, K. Skadron, S. Gurumurthi, R. J. Ribando and M. R. Stan, "Differentiating the Roles of IR Measurement and Simulation for Power and Temperature-aware Design", *ISPASS* 2009, pp. 1-10.

[47] C.-H. Hsu, H.-Y. Chen and Y.-W. Chang, "Multi-layer Global Routing Considering Via and Wire Capacities", *ICCAD* 2008, pp. 350-355.

[48] M.-K. Hsu, S. Chou, T.-H. Lin and Y.-W. Chang, "Routability-driven Analytical Placement for Mixed-size Circuit Designs", *ICCAD* 2011, pp. 80-84.

[49] L. Hsu, R. Iyer, S. Makineni, S. Reinhardt and D. Newell, "Exploring the Cache Design Space for Large Scale CMPs," *Computer Architecture News* 2005, pp. 24-33.

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

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

[52] International Technology Roadmap for Semiconductors (ITRS). `http://www.itrs.net`.

[53] International Technology Roadmap for Semiconductors.
`http://www.itrs.net/Links/2010ITRS/Home2010.htm`

[54] D. Jariwala and J. Lillis, "RBI: Simultaneous Placement and Routing Optimization Technique", *TCAD* 26(1) (2007), pp. 127-141.

[55] K. Jeong and A. B. Kahng, "A Power-constrained MPU Roadmap for the International Technology Roadmap for Semiconductors (ITRS)", *ISOCC* 2009, pp. 49-52.

[56] K. Jeong and A. B. Kahng, "Toward PDN Resource Estimation: A Law of General Power Density", *SLIP* 2011, pp. 1-6.

[57] K. Jeong, A. B. Kahng and C. J. Progler, "New Yield-aware Mask Strategies", *PMJ* 2011, pp. 80810P-1–80810P-12.

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

[59] A. B. Kahng, J. Lienig, I. L. Markov and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*, Springer, 2011.

[60] A. B. Kahng, S. Mantik and D. Stroobandt, "Toward Accurate Models of Achievable Routing", *TCAD* 8(6) (2001), pp. 648-659.

[61] A. B. Kahng and X. Xu, "Accurate Pseudo-constructive Wirelength and Congestion Estimation" *SLIP* 2003, pp. 61-68.

[62] R. Karp, “Complexity of Computer Computations”, *Reducibility Among Combinatorial Problems*, New York: Plenum, 1972.

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

[64] G. Katti, M. Stucchi, K. D. Meyer and W. Dehaene, “Electrical Modeling and Characterization of Through Silicon Via for Three-dimensional ICs”, *Electron Devices* 57(1) (2010), pp. 256-262.

[65] J. Hu*, M.-C. Kim*, D.-J. Lee and I. L. Markov, “A SimPLR Method for Routability-driven Placement”, *ICCAD* 2011, pp. 67-73. **Equal Contribution*

[66] M.-C. Kim, D.-J. Lee and I. L. Markov, “SimPL: An Effective Placement Algorithm”, *ICCAD* 2010, pp. 649-656.

[67] M.-C. Kim, D.-J. Lee and I. L. Markov, “SimPL: An Effective Placement Algorithm”, *TCAD* 31(1) (2012), pp. 50-60.

[68] M.-C. Kim and I. L. Markov, “ComPLx: A Competitive Primal-dual Lagrange Optimization for Global Placement”, *DAC* 2012, pp. 747-752.

[69] B. Korte, D. Rautenbach and J. Vygen, “BonnTools: Mathematical Innovation for Layout and Timing Closure of Systems on a Chip”, *Proc. IEEE* 95(3) (2007), pp. 555-572.

[70] T.-H. Lee, Y.-J. Chang and T.-C. Wang, “An Enhanced Global Router with Consideration of General Layer Directives”, *ISPD* 2011, pp. 53-60.

[71] S. Lee and M. D. F. Wong, “Timing-driven Routing for FPGAs Based on Lagrangian Relaxation”, *TCAD*, 22(4) (2003), pp. 506-510.

[72] T.-H. Lee and T.-C. Wang, “Congestion-constrained Layer Assignment for Via Minimization in Global Routing”, *TCAD* 27(9) (2008), pp. 1643-1656.

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

[74] Z. Li, C. J. Alpert, G.-J. Nam, C. C. N. Sze, N. Viswanathan and N. Y. Zhou, “Guiding a Physical Design Closure System to Produce Easier-to-route Designs with More Predictable Timing”, *DAC* 2012, pp. 465-470.

[75] C. Li, M. Xie, C.-K. Koh, J. Cong and P. H. Madden, “Routability-driven Placement and White Space Allocation”, *ICCAD* 2004, pp. 394-401.

[76] Z. Li, W. Wu and X. Hong, “Congestion Driven Incremental Placement Algorithm for Standard Cell Layout”, *ASP-DAC* 2003, pp. 723-728.

[77] W.-H. Liu, W.-C. Kao, Y.-L. Li and K.-Y. Chao, "Multi-threaded Collision-aware Global Routing with Bounded-length Maze Routing", *DAC* 2010, pp. 200-205.

[78] W.-H. Liu, Y.-L. Li and C.-K. Kok, "A Fast Maze-free Routing Congestion Estimator With Hybrid Unilateral Monotonic Routing", *ICCAD* 2012, pp. 713-719.

[79] H. Van Marck, D. Stroobandt and J. Van Campenhout, "Towards an Extension of Rent's Rule for Describing Local Variations in Interconnect Complexity", *International Conf. for Young Computer Scientists* (1994), pp. 136-141.

[80] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-based Performance-driven Router for FPGAs", *FPGA* 1995, pp. 111-117.

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

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

[83] G.-J. Nam, F. Aloul, K. A. Sakallah and R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints", *ISPD* 2001, pp. 222-227.

[84] G.-J. Nam, C. C. N. Sze and M. C. Yildiz, "The ISPD Global Routing Benchmark Suite", *ISPD* 2008, pp. 156-159. `http://www.sigda.org/ispd2008/contests/ispd08rc.html`

[85] R. H. J. M. Otten and R. K. Brayton, "Planning for Performance", *DAC* 1998, pp. 122-127.

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

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

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

[89] M. Pan, N. Viswanathan and C. C. N. Chu, "An Efficient and Effective Detailed Placement Algorithm", *ICCAD* 2005, pp. 48-55.

[90] M. Pan, Y. Xu, Y. Zhang and C. Chu, "FastRoute: An Efficient and High-quality Global Router", *VLSI Design* 2012, 18 pages.

[91] P. N. Parakh, R. B. Brown and K. A. Sakallah, "Congestion Driven Quadratic Placement", *DAC* 1998, pp. 275-278.

[92] S. K. Raman, V. Pentkovski and J. Keshava, "Implementing Streaming SIMD Extensions on the Pentium III Processor", *Proc. Micro* 20(4) (2000), pp. 47-57.

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

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

[95] J. A. Roy, N. Viswanathan, G.-J. Nam, C. J. Alpert and I. L. Markov, "CRISP: Congestion Reduction by Iterated Spreading during Placement", *ICCAD* 2009, pp. 357-362.

[96] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, 2003.

[97] P. Saxena, N. Menezes, P. Cocchini and D. A. Kirkpatrick, "The Scaling Challenge: Can Correct-by-Construction Design Help?", *ISPD* 2003, pp. 51-58.

[98] N. Selvakkumaran, P. N. Parakh and G. Karypis, "Perimeter-degree: A Priori Metric for Directly Measuring and Homogenizing Interconnection Complexity in Multilevel Placement", *SLIP* 2003, pp. 53-59.

[99] W. Y. Seung, W. Y. Dae, H. K. Jae, M. Padmanathan and F. Carson, "3D TSV Processes and its Assembly/Packaging Technology", *3DIC* 2009, pp. 1-5.

[100] H. Shojaei, A. Davoodi and J. Linderoth, "Congestion Analysis for Global Routing via Integer Programming", *ICCAD* 2011, pp. 256-262.

[101] G. Sigl, K.Doll and F.Johannes,"Analytical Placement: A Linear or a Quadratic Objective Function?" *DAC*1991, pp. 427-432.

[102] P. Spindler and F. M. Johannes, "Fast and Accurate Routing Demand Estimation for Efficient Routability-driven Placement", *DATE* 2007, pp. 1226-1231.

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

[104] Wafer Stacking with Super-contacts. `http://www.tezzaron.com`

[105] K. Tsota, C. Koh and V. Balakrishnan, "Guiding Global Placement with Wire Density", *ICCAD* 2008, pp. 212-217.

[106] TSMC: Silicon Success.
`http://www.tsmc.com/download/enliterature/html-newsletter/September03/InDepth/index.html`

[107] N. Viswanathan, IBM, *personal communication*, November 2011.

[108] N. Viswanathan, C. J. Alpert, C. C. N. Sze, Z. Li, G.-J. Nam and J. A. Roy, "The ISPD-2011 Routability-driven Placement Contest and Benchmark Suite", *ISPD* 2011, pp. 141-146.

[109] N. Viswanathan, C. J. Alpert, C. C. N. Sze, Z. Li and Y. Wei, "The DAC 2012 Routability-driven Placement Contest and Benchmark Suite", *DAC* 2012, pp. 774-782.

[110] N. Viswanathan, C. J. Alpert, C. Sze, Z. Li and Y. Wei, "ICCAD-2012 CAD Contest in Design Hierarchy Aware Routability-driven Placement and Benchmark Suite", *ICCAD* 2012, pp. 345-348. `http://cad_contest.cs.nctu.edu.tw/CAD-contest-at-ICCAD2012/problems/p2/p2.html`

[111] N. Viswanathan, M. Pan and C. C. N. Chu, "FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion Control", *ASP-DAC* 2007, pp. 135-140.

[112] M. Wang and M. Sarrafzadeh, "On the Behaviour of Congestion Minimization During Placement", *ISPD* 1999, pp. 145-150.

[113] M. Wang and M. Sarrafzadeh, "Model and Minimization of Routing Congestion", *ASP-DAC* 2000, pp. 185-190.

[114] M. Wang, X. Yang, K. Eguro and M. Sarrafzadeh, "Multicenter Congestion Estimation and Minimization During Placement", *ISPD* 2000, pp. 147-152.

[115] M. Wang, X. Yang and M. Sarrafzadeh, "Congestion Minimization During Placement", *TCAD* 19(10) (2000), pp. 1140-1148.

[116] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. N. Reddy, A. D. Huber, G. E. Terez, D. Keller and S. S. Sapatnekar, "GLARE: Global and Local Wiring Aware Routability Evaluation", *DAC* 2012, pp. 768-773.

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

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

[119] T.-H. Wu, A. Davoodi and J. T. Linderoth, "A Parallel Integer Programming Approach to Global Routing", *DAC* 2010, pp. 194-199.

[120] H. Xu, R. Rutenbar and K. A. Sakallah, "sub-SAT: A Formulation for Relaxed Boolean Satisfiability with Applications in Routing", *ISPD* 2002, pp. 182-187.

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

[122] Y. Xu and C. C. N. Chu, "MGR: Multi-level Global Router", *ICCAD* 2011, pp. 250-255.

[123] X. Yang, B.-K. Choi and M. Sarrafzadeh, "Routability-driven White Space Allocation for Fixed-die Standard-cell Placement", *TCAD* 22(4) (2003), pp. 410-419.

[124] X. Yang, R. Kastner and M. Sarrafzadeh, “Congestion Estimation During Top-down Placement”, *TCAD* 21(1) (2002), pp. 72-80.

[125] J. Y. Yen, “An Algorithm for Finding Shortest Routes From All Source Nodes to a Given Destination in General Networks”, *Proc. Quarterly of Applied Mathematics* 27 (1970), pp.526-530.

[126] Y. Zhang and C. C. N. Chu, “CROP: Fast and Effective Congestion Refinement of Placement”, *ICCAD* 2009, pp. 344-350.

[127] Y. Zhang and C. Chu, “GDRouter: Interleaved Global Routing and Detailed Routing for Ultimate Routability”, *DAC* 2012, pp. 597-602.

[128] K. Zhong and S. Dutt, “Algorithms for Simultaneous Satisfaction of Multiple Constraints and Objective Optimization in a Placement Flow with Application to Congestion Control”, *DAC* 2002, pp. 854-859.