High-performance Placement and Routing for the Nanometer Scale

by

Jarrod Alexander Roy

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
2009

Doctoral Committee:

Associate Professor Igor L. Markov, Chair
Professor David Blaauw
Professor John P. Hayes
Associate Professor Kevin J. Compton
Associate Professor Dennis M. Sylvester

To my family and friends

ACKNOWLEDGEMENTS

I am grateful to my advisor, Professor Igor Markov, for taking a chance on me when I was a Master's student and supporting me throughout my Ph.D. He has been an unending source of ideas and advice in research and insightful comments when writing all of our papers. I am also thankful to my thesis committee members Professor John Hayes, Professor Kevin Compton, Professor David Blaauw and Professor Dennis Sylvester for reminding me to always think about solving difficult problems from both theoretical and practical points of view. I'd also like to thank Professor Farinaz Koushanfar for her collaboration on our work on hardware piracy and intellectual property protection.

I'd like to thank David Papa for all of our debates on subjects ranging from proper programming practice to finance and for convincing me to get active and lose weight. I want to thank Jin Hu for taking up and improving upon my global routing work. I would be remiss if I did not thank my group at IBM Austin Research Lab who helped with the work I needed to complete my thesis including Charles Alpert, Gi-Joon Nam, Natarajan Viswanathan, Michael Moffitt, Cliff Sze, Zhuo Li and Nancy Zhou. I'm also appreciative of the inordinate hours spent with Robert Barry, George Viamontes, Aaron Ng, Jeff Hao, David Papa and Jin Hu playing video games (WoW, Diablo II, Guild Wars, Guitar Hero, Rock Band, ...) which helped to preserve my sanity. I would like to thank all of the other brilliant people at the University of Michigan that I was fortunate enough to befriend including Kai-Hui Chang, Saurabh Adya, Smita Krishnaswamy, James Lu, Hector Garcia, Andrew DeOrio, Ilya Wagner, Joseph Greathouse and Stephen Plaza. I'd also like to

thank my friends from Carnegie Mellon University who have kept in touch over the years including Ben Burrington, Marc Giogilo and Doug Houston.

Without my family, none of this could have been possible. I thank my sister, Jinger, for moving to Michigan for a few months, convincing me to adopt a cat and keeping me cultured. Lastly and far from least, I'd like to thank my parents, Julius and Diane, for bringing me to early morning MathCounts practices before school; buying my first 80386 which introduced me to the world of computing; letting me go off to college and graduate school 1000 miles from home; moving up to Michigan for a year; and always being there for me.

PREFACE

Modern semiconductor manufacturing facilitates single-chip electronic systems that only five years ago required ten to twenty chips. Naturally, design complexity has grown within this period. In contrast to this growth, it is becoming common in the industry to limit design team size which places a heavier burden on design automation tools.

Our work identifies new objectives, constraints and concerns in the physical design of systems-on-chip, and develops new computational techniques to address them. In addition to faster and more relevant design optimizations, we demonstrate that traditional design flows based on "separation of concerns" produce unnecessarily suboptimal layouts. We develop new integrated optimizations that streamline traditional chains of loosely-linked design tools. In particular, we bridge the gap between mixed-size placement and routing by updating the objective of global and detail placement to a more accurate estimate of routed wirelength. To this we add sophisticated whitespace allocation, and the combination provides increased routability, faster routing, shorter routed wirelength, and the best via counts of published techniques. To further improve post-routing design metrics, we present new global routing techniques based on Discrete Lagrange Multipliers (DLM) which produce the best routed wirelength results on recent benchmarks. Our work culminates in the integration of our routing techniques within an incremental placement flow to improve detailed routing solutions, shrink die sizes and reduce total chip cost.

Not only do our techniques improve the quality and cost of designs, but also simplify design automation software implementation in many cases. Ultimately, we reduce the time needed for design closure through improved tool fidelity and the use of our incremental techniques for placement and routing.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

PART I

Background

CHAPTER I

VLSI Physical Design at the Nanometer Scale

1.1 Challenges in VLSI Physical Design

Very Large Scale Integration (VLSI) circuit design has reached unheralded scope, and projections show a steady increase in size and complexity as the industry moves further into the nanometer scale. State-of-the-art systems-on-chip routinely have tens of millions of standard cells and signal nets. Therefore, powerful and scalable techniques for placement and routing are crucial to enable designs to reach *design closure*, i.e., satisfying all constraints and objectives such as power limits, clock-cycle times and yield targets. Without powerful techniques, designs often require multiple iterations which include time-consuming manual changes performed by designers.

Recently there has been much interest in estimating the amount of improvement left in

placement optimization [30]. The gap between *optimal* and *practically achievable* solutions is usually explained by the difficulty of optimization and shortcomings of individual algorithms. We point out another major source of sub-optimality in physical design — *minimizing wrong objective functions*, whether optimally or not.

In a modern nanoscale design flow, circuit and system optimizations must interact with physical aspects of the design. For example, improvements in timing and power may require replacing large modules with variants that have different power/delay trade-offs, shapes and connectivity. New logic may be added late in the design flow, subject to interconnect optimization. To support such flexibility in design flows, a robust system for performing Engineering Change Orders (ECOs) is necessary, but no such system has been previously detailed in the literature. In fact, there is considerable disagreement as to what sorts of incremental design changes must be supported by an ECO tool.

In order to accurately evaluate timing and other design characteristics, routing must take place to connect design components with wire. VLSI routing is an active area of research and development with a growing body of literature [7, 40, 64, 109, 110]. Current efforts in routing are motivated primarily by challenges present at the nanometer scale. Such challenges include an explosion in design rules which must be obeyed to promote manufacurability and yield and satisfying density constraints to prevent both uneven chip wear during chemical mechanical polishing (CMP) [41] and the impact of capacitance on interconnect delay. To route designs effectively while solving these problems, powerful, efficient (both in terms of CPU and memory resources), and flexible routing infrastructure must be developed.

1.2 Our Approach

We identify new objectives, constraints and concerns in VLSI physical design, and develop new computational techniques to address them. Our goal is to reduce the number of iterations necessary to achieve nanometer design closure by improving the solution quality and robustness of tools as well as making optimizations more consistent across different "point" tools in a modern design flow. Therefore, our work spans the usually distinct tasks of global placement, detailed placement and global routing.

Traditional VLSI placement tools minimize the *half-perimeter wirelength (HPWL)* of a design rather than more relevant metrics such as timing or more accurate predictors of routed wirelength. HPWL is easy to compute, but badly underestimates routed wirelength for nets with more than a few pins. It is well-known that routers build Steiner trees to produce the best results, so we examine the minimization of Steiner-tree wirelength in global and detailed placement. Our work leverages existing research on Rectilinear Steiner Minimal Tree (RSMT) construction, and our placer ROOSTER (Rigorous Optimization Of Steiner Trees Eases Routing) is the first in the literature to target Steiner-tree length in global placement. By minimizing Steiner-tree length in global and detailed placement combined with routability-driven whitespace allocation, ROOSTER outperforms the vast majority of published results in routed wirelength and routing time. More importantly at the nanometer scale, ROOSTER produces best published results in via counts on a wide variety of publicly available benchmarks.

To preserve design properties across optimizations, Engineering Change Orders (ECOs) and design iterations, a fast and robust incremental placement technique is necessary. We

build upon the well-known, robust and scalable min-cut placement framework to perform incremental placement. In contrast with existing stand-alone tools that offer poor interfaces to the design flow and cannot handle a full range of modern VLSI layouts, our incremental placer ECO-system reliably handles fixed objects and movable macros in instances with widely varying amounts of whitespace. It detects geometric regions and sections of the netlist that require modification and applies an adequate amount of change in each case. Given a reasonable initial placement, it applies minimal changes, but is capable of replacing large regions to handle pathological cases. ECO-system can be used in the range from high-level synthesis, to physical synthesis and detailed placement. ECO-system is many times faster than a global placer, increases wirelength only slightly, and has minimal impact on timing.

To tackle the wide range of issues in VLSI routing at the nanometer scale, we build a global routing framework based on a new computational technique Discrete Lagrange Multipliers (DLM) which has been validated for two- and three-dimensional routing of ASICs with millions of nets. Our framework, which we call FGR (Fairly Good Router), outperforms the best results from the ISPD 2007 Global Routing Contest, as well as previous literature, in terms of route completion, runtime and total wirelength.

Working on both placement and routing offers a unique opportunity to improve the design closure process. Even though placement and routing are usually seen as separate tasks with differing goals, we contend that they should be treated as a single step. Thus, we incorporate global routing operations into placement with CRISP (Congestion Reduction by Iterated Spreading during Placement). CRISP uses actual congestion information from

a global router rather than estimates from probabilistic congestion maps. CRISP closes the gap between placement and routing in modern design flows, improving overall design quality and reducing time for design closure.

1.3 Organization of the Thesis

The remainder of this dissertation is organized as follows. Part I outlines relevant background in VLSI placement in Chapter II and routing in Chapter III. Part II covers our work in VLSI placement. Chapter IV describes methods for accurately controlling whitespace allocation during placement. Chapter V introduces the ROOSTER placer and details our techniques for improving routability in placement without the aid of a global router. Chapter VI presents our work on incremental placement called ECO-system. Part III describes the framework we have developed for global routing, FGR, in Chapter VII and proposes several extensions. Chapter VIII describes FGR's efficient data structures, extending FGR's discrete Lagrange formulation, and improving FGR's runtime and solution quality. Part IV presents integration of global routing into modern design flows. Chapter IX describes the CRISP incremental routability enhancing technique which interleaves calls to a global router with placement manipulation and spreading to improve both global and detailed routability. Part V summarizes the dissertation and discusses directions for future work in Chapter X.

CHAPTER II

State of the Art in Partitioning-based VLSI Placement

Using balanced min-cut partitioning in placement was described by Breuer in 1977 [19] and was employed by both IBM and Bell Labs at least ten years earlier. Min-cut placers use a scalable and extensible divide-and-conquer algorithmic framework and tend to produce routable placements [21]. This success has motivated much research in efficient partitioning algorithms []. Recent work in partitioning-based placement offers extensions to block placement and large-scale mixed-size placement [37, 50, 115], and robust incremental placement [117].

Over the years, partitioning-based placement has seen many revisions and enhancements, but the underlying framework (illustrated in Figure 2.1) remains much the same. Top-down partitioning-based placement algorithms seek to decompose a given placement instance into smaller instances by sub-dividing the placement region, assigning modules to subregions and cutting the netlist hypergraph [19,58]. The top-down placement process can be viewed as a sequence of passes where each pass examines all bins and divides some of them into smaller bins. Most commonly the division step is accomplished with balanced min-cut partitioning that minimizes the number of signal nets connecting modules in mul-

ALGORITHM 2.1: Partitioning-based placement

```
    ▷ Input: queue of placement bins Q, netlist to place
    ▷ Output: placements of all the movable objects in netlist
1   while (EMPTY(Q) = FALSE)
2       do bin ← DEQUEUE(Q)
3          if (DETERMINEBINSIZE(bin) = SMALL)
4            then CALLENDCASEPLACER(bin)
5            else direction ← CHOOSECUTLINEDIRECTION(bin)
6                 cutline ← CHOOSECUTLINEPOSITION(bin, direction)
7                 graph ← BUILDPARTITIONINGGRAPH(bin, cutline, netlist)
8                 childBins ← CALLPARTITIONER(bin, cutline, graph)
9                 ENQUEUE(Q, childBins)
```

Figure 2.1: Traditional top-down partitioning-based placement.

tiple regions [19]. These techniques leverage well-understood and scalable algorithms for hypergraph partitioning and typically lead to routable placements [21].

This chapter details partitioning-based placement techniques and illustrates how they are used in state-of-the-art min-cut placers. Section 2.1 introduces the basic framework for min-cut partitioning-based placement. Next, Section 2.2 presents recent enhancements to min-cut placement. Section 2.3 describes adapting partitioning based methods to mixed-size placement. Lastly, Section 2.4 discusses state-of-the-art min-cut placers such as Dragon [129, 130, 143], FengShui [6, 86], NTUPlace2 [74] and Capo [107, 115–117, 119, 120] and illustrates how they differ from the min-cut framework and each other.

2.1 Top-down Partitioning Framework for Placement

The underlying min-cut partitioning framework remains mostly the same since Breuer's work in 1977 [19], and is illustrated in Figures 2.1 and 2.2. The placement region is represented by a series of *placement bins* which represent (i) a placement region with allowed module locations (*sites*), (ii) a collection of circuit modules to be placed in this region,

(iii) all signal nets incident to the modules in the region, and (iv) fixed cells and pins outside the region that are adjacent to modules in the region (*terminals*).

Min-cut partitioning-based placers generally proceed by dividing the netlist and placement area into successively smaller pieces until the pieces are small enough to be handled efficiently by optimal end-case placers [23]. State-of-the-art placers generally use a wide range of hypergraph partitioning techniques to best fit partitioning problem size — optimal (branch-and-bound [23]), middle-range (Fiduccia-Mattheyses [59]) and large-scale (multi-level Fiduccia-Mattheyses [22, 84]). Min-cut placement is highly scalable (due in large part to algorithmic advances in min-cut partitioning [22, 59, 84]) and typically produces routable placements.

In this section, we introduce topics relevant to top-down partitioning-based placement that must be addressed by all modern min-cut placers. Specifically we discuss terminal propagation, bipartitioning vs. multi-way partitioning, cut-line selection and whitespace (or free space) allocation.

2.1.1 Terminal Propagation

Proper handling of terminals is essential to the success of top-down placement approaches [23, 58, 68, 128]. When a particular placement bin is split into multiple sub-bins, some of the cells inside may be tightly connected to cells outside of the bin. Ignoring such connections can adversely affect the quality of a placement since these connections can account for significant amounts of wirelength. On the other hand, these terminals are irrelevant to the classic partitioning formulation as they cannot be freely assigned to partitions. A compromise is possible by using an extended formulation of "partitioning with

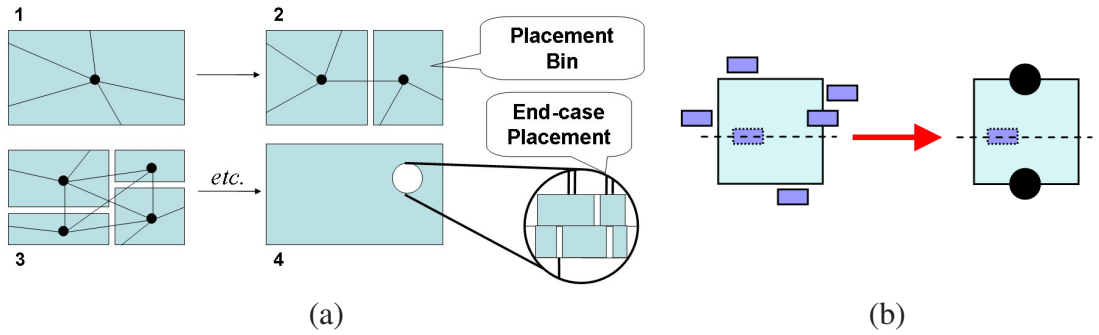


Figure 2.2: (a) Top-down bisection-based placement. The placement area and netlist are successively divided into placement bins until the bins are small enough for end-case placement. (b) Terminal propagation is an important enhancement to top-down bisection-based placement. The net has five fixed terminals (four above and one below the cut-line) and movable cells which are represented by the cell with a dashed outline. The fixed terminals above the cut-line are propagated to the black circle at the top of the bin; the fixed terminal below the cut-line is propagated to the black circle below the cut-line; and movable cells remain unpropagated. Note that the net is *inessential* since terminals are propagated to both sides of the cut-line [116].

fixed terminals", where the terminals are considered to be fixed in ("propagated to") one or more partitions, and assigned zero areas (original areas are ignored) [58]. Nets which are propagated to both partitions in bipartitioning are considered "inessential" since they will always be cut and can be safely removed from the partitioning instance to improve runtime [23]. Terminal propagation is typically driven by geometric proximity of terminals to bins. Figure 2.2 depicts terminal propagation for a net with fixed terminals. This net is inessential as it has terminals propagated to both sides of the cut-line.

2.1.2 Bipartitioning versus Multi-way Partitioning

In his seminal work on min-cut placement, Breuer introduced two forms of recursive min-cut placement which he called *slice/bisection* and *quadrature* [19]. The style of min-cut placement most commonly used today, which is known as *bipartitioning* or *bisection*, has grown from the quadrature technique which advocated the use of horizontal and

vertical cuts; the slice/bisection technique used only horizontal cuts and exhibited worse performance than quadrature [19].

Since that time, horizontal and vertical cut-lines have been standard in all placement techniques, but there has been debate as to whether there should be an ordering to the cuts (i.e., horizontally bisect a bin then vertically bisect its children as in quadrature [19]) or both cuts should be done simultaneously as in *quadrisection* [128]. Quadrisection has been shown to allow for the optimization of techniques other than min-cut (such as minimal spanning tree length [68]), but terminal propagation is more complex when splitting a bin into four child bins instead of two. Also, bisection can simulate quadrisection with added flexibility in cut-line selection and shifting (see Section 2.1.3) [115]. There are currently no known methods that use greater than 4-way partitioning and the vast majority of partitioning-based placement techniques involve min-cut bipartitioning.

2.1.3 Cut-line Selection and Shifting

Breuer studied two types of cut-line direction selection techniques and found that alternating cut-line directions from layer to layer produced better half-perimeter wirelength (HPWL) than using only horizontal cuts [19]. The authors of [131] studied this phenomenon further by testing 64 cut-line direction sequences. Their experiments did not find that the two cut-sequences that alternate at each layer were the best, but did find that long sequences of cuts in the same direction during placement was detrimental to performance [131]. The authors of [145] developed a dynamic programming technique to choose optimal cut sequences for partitioning based placement, but also found that nearly optimal cut sequences could be determined from the aspect ratio of the bin to be split.

After cut-line direction is chosen, partitioning-based placers generally choose the cut-line that best splits a placement bin in half in the desired direction. Cut-lines are generally aligned to placement row and site boundaries to ease the assignment of standard-cells to rows near the end of global placement [21]. After a bin is partitioned, the initial cut-line may be moved, or shifted, in order to satisfy other objectives such as whitespace allocation or congestion reduction.

2.1.4 Whitespace Allocation

Management of *whitespace* (also known as *free space*) is a key issue in physical design as it has a profound effect on design routability, bufferability, spare-cell insertion, etc. Applications of whitespace allocation are covered in greater detail Chapters IV-VI and IX. The amount of whitespace in a design is the difference between the total placeable area in a design and the total movable cell area in the design. A natural scheme for managing whitespace in top-down placement, uniform whitespace allocation, was introduced and analyzed in [24]. Let a placement bin which is going to be partitioned have *site area* S, *cell area* C, *absolute whitespace* $W = \max\{S - C, 0\}$ and *relative whitespace* $w = W/S$. A bipartitioning divides the bin into two child bins with *site areas* S_0 and S_1 such that $S_0 + S_1 = S$ and *cell areas* C_0 and C_1 such that $C_0 + C_1 = C$. A partitioner is given cell area targets T_0 and T_1 as well as a tolerance τ for a particular bipartitioning instance. In many cases of bipartitioning, $T_0 = T_1 = \frac{C}{2}$, but this is not always true [10]. τ defines the maximum percentage by which C_0 and C_1 are allowed to differ from T_0 and T_1, respectively.

The work in [24] bases its whitespace allocation techniques on *whitespace deteriora-*

tion: the phenomenon that discreteness in partitioning and placement does not allow for exact uniform whitespace distribution. The whitespace deterioration for a bipartitioning is the largest α, such that each child bin has at least αw relative whitespace. Assuming non-zero relative whitespace in the placement bin, α should be restricted such that $0 \leq \alpha \leq 1$ [24]. The authors note that $\alpha = 1$ may be overly restrictive in practice because it induces zero tolerance on the partitioning instance but $\alpha = 0$ may not be restrictive enough as it allows for child bins with zero whitespace, which can improve wirelength but impair routability [24].

For a given block, the feasible ranges for partition capacities are determined by α. The partitioning tolerance τ for splitting a block with relative whitespace w is $\frac{(1-\alpha)w}{1-w}$. The challenge is to determine a proper value for α. First assume that a bin is to be partitioned horizontally n times more during the placement process. n can be calculated as $\lceil \log_2 R \rceil$ where R is the number of rows in the placement bin. Assuming end-case bins have $\alpha = 0$ since they are not further partitioned, $\overline{w}$, the relative whitespace of an end-case bin, is determined to be $\frac{\overline{\tau}}{\overline{\tau}+1}$ where $\overline{\tau}$ is the tolerance of partitioning in the end-case bin [24].

Assuming that α remains the same during all partitioning of the given bin gives a simple derivation of $\alpha = \sqrt[n]{\frac{\overline{w}}{w}}$. A more practical calculation assumes instead that τ remains the same over all partitionings. This leads to $\tau = \sqrt[n]{\frac{1-\overline{w}}{1-w}} - 1$. $\overline{w}$ can be eliminated from the equation for τ and a closed form for α is derived: $\alpha = \frac{\sqrt[n+1]{1-w} - (1-w)}{w(\sqrt[n+1]{1-w})}$ [24].

Free Cell Addition. One method of non-uniform whitespace allocation in placement, was presented in [5]. To achieve a non-uniform allocation of whitespace, free cells (standard cells that have no connections in the netlist) are added to the design which is then

placed using uniform whitespace allocation. Care must be taken not to add too many cells to the design which can complicate the work of many placement algorithms, increasing interconnect length or leading to overlapping circuit modules [50].

2.2 Enhancements to the Min-cut Framework

This section describes several techniques which are recent improvements to the min-cut partitioning-based framework presented in Section 2.1. These techniques range from fairly simple yet effective techniques such as repartitioning and placement feedback to changes in the optimization goals of min-cut placement, as in weighted net-cut.

2.2.1 Better Results Through Additional Partitioning

Huang and Kahng introduced two techniques for improving the results of quadrisection based placement known as cycling and overlapping [68]. Cycling is a technique whereby results are improved by partitioning every placement bin multiple times for each layer [68]. After all bins are split for the first time in a layer of placement, a new round of partitioning on the same bins is done using the results of the previous round for terminal propagation. These additional rounds of partitioning are repeated until there is no further improvement of a cost function [68]. A similar type of technique was presented for min-cut bisection called *placement feedback*. In placement feedback, bins are partitioned multiple times, without requiring steady improvement in wirelength, to achieve more consistent terminal propagation [79].

Placement feedback serves to reduce the number of ambiguously propagated terminals. Ambiguity in terminal propagation arises when a terminal is nearly equidistant from

the centers of the child bins of the bin being partitioned. In such cases it is unclear as to what side of the cut-line the terminal should be propagated. Traditional choices for such terminals are to propagate them to both sides or neither side of the cut-line in fear of making a poor decision [79]. Ambiguously propagated terminals introduce non-determinism into min-cut placement as they may be propagated differently based on the order in which placement bins are processed [79].

To reduce the number of ambiguously propagated terminals, placement feedback repeats each layer of partitioning n times. Each successive round of partitioning uses the resulting locations from the previous partitioning for terminal propagation. The first round of partitioning for a particular layer may have ambiguous terminals, but the second and later rounds will have reduced numbers of ambiguous terminals making terminal propagation more robust [79]. Empirical results show that placement feedback is effective in reducing HPWL, routed wirelength and via count [79].

The technique of overlapping also involves additional partitioning calls during placement [68]. While doing cycling in quadrisection, pieces of neighboring bins can be coalesced into a new bin and split to improve solution quality [68]. Brenner and Rohe introduced a similar technique called repartitioning which was designed to reduce congestion [16]. After partitioning, congestion was estimated in the placement bins of the design. Using this congestion data, new partitioning problems were formulated with all neighbors of a congested area. Solving these new partitioning problems would spread congestion to neighboring areas of the placement while possibly incurring an increase in net length [16].

Capo [107, 115–117, 119, 120] repartitions bins similarly, but for the improvement of

HPWL. After the initial solution of a partitioning problem is returned from a min-cut partitioner, Capo has the option of shifting the cut-line to fulfill whatever whitespace requirements may be asked of it. A shift of the cut-line, though, represents a change in the partitioning problem formulation (as the initial partitioning problem was built assuming a different cut-line which can have a significant effect on terminal propagation). Thus, the partitioning problem is rebuilt with the new cut-line location and solved again to improve wirelength. The repartitioning does not come with a significant runtime penalty because the initial partitioning solution is reused and modified by flat passes of a Fiduccia-Mattheyses [59] partitioner.

2.2.2 Fractional Cut

When a placement bin is split with a vertical cut-line, there are generally many possible cut-lines that can split the bin roughly equally since the size of sites in row-based placement is generally small. On the other hand, row heights are generally non-trivial as compared to the height of the core placement area. Since standard cells are ultimately placed in rows, most min-cut placers choose to align cut-lines to row boundaries [21]. The authors of [6] note that this causes the "narrow region" problem which leads to instability in min-cut placement.

The "narrow region" problem becomes an issue when bins become tall and narrow. In such cases, a narrow bin may contain enough area to fit a group of standard cells, but it may not be possible to assign cells to legal locations within the bin; or the number of legal solutions is so small that net-cut is artificially increased as a result [6]. Take for example a placement bin that encompasses two adjacent rows, each four units in length. If we have

one cell with length five units and another with length two units, there is no way to legally place them in the rows, yet total area constraints are satisfied.

To remedy this situation, the authors of [6] propose using a "fractional cut": a horizontal cut-line that is allowed to pass through a fraction of a row. As horizontal cut-lines do not necessarily align with rows, cells must be assigned to rows before optimal end-case (typically single-row) placers can be used [6]. To legalize the placement, one proceeds on a row-by-row basis. Each cell is tentatively assigned to a preferred height in the placement: the center of its placement bin. Starting with the top-most row, cells are assigned to rows so as to minimize the cost of assigning cells. If a cell is assigned to the current row, its cost is the squared distance from its preferred position to the current row. If a cell is not assigned to the current row, its cost is the squared distance from its preferred position to the next lower row [6]. After all cells are assigned to rows, they are sorted by their x coordinates and packed in rows to remove any overlaps. The assignment of cells to rows is achieved efficiently by a dynamic programming formulation [6]. Experimental results show considerable improvements in terms of HPWL reduction in placement, but packing of cells in rows does not generally produce routable placements [116].

2.2.3 Analytical Constraint Generation

The authors of [10] note that min-cut placement techniques are effective at reducing HPWL of designs that are heavily constrained in terms of whitespace, but do not perform nearly as well as analytical techniques when there are large amounts of whitespace. The authors suggest that one reason for the discrepancy is that min-cut placers generally try to divide placement bins exactly in half with a relatively small tolerance. This tends to spread

cell area roughly uniformly across the core area. Increasing the tolerance for partitioning a bin can allow for less uniformity in placement and lower HPWL due to tighter packing, but still does not reproduce the performance of analytical techniques [10].

To improve the HPWL performance of min-cut placement techniques on designs with large amounts of whitespace (which are becoming increasingly popular in real-world designs), while still retaining the good performance of min-cut techniques when there is limited whitespace, the authors of [10] suggest integrating analytical techniques and min-cut techniques. Before constructing a partitioning instance for a given placement bin, an analytical technique is run on the objects in the bin to minimize their quadratic wirelength [10]. Next, the center of mass of the placement of the objects of the bin is calculated. This points to roughly where the objects should go to reduce their wirelength. Then one constructs a rectangle having the same aspect ratio as the placement bin and having the same area as the total area of movable objects in the bin. Let A be the total area of cells in the bin, H be the height of the bin and W the width of the bin. The height and width of such a rectangle can be calculated as follows: rectangle height $RH = \sqrt{\frac{A*H}{W}}$ and rectangle width $RW = \sqrt{\frac{A*W}{H}}$ [10]. One centers this rectangle at the center of mass of the analytical placement and intersects the rectangle with the proposed cut-line of the bin. The amount of area of the rectangle that falls on either side of the cut-line is used as a target for min-cut partitioning [10]. As most min-cut partitioners chose to split cell area equally, this is a significant departure from traditional min-cut placement.

Empirical results suggest that analytical constraint generation (ACG) is effective at improving the performance of min-cut placement on designs with large amounts of whites-

pace while retaining the good performance of min-cut placers on constrained designs while not impairing the routability of designs [10]. This performance comes at the cost of approximately 28% more runtime [10].

2.2.4 Better Modeling of HPWL by Partitioning

It is well-known that the min-cut objective in partitioning does not accurately represent the wirelength objective of placement [68, 123]. Optimizing HPWL and other objectives directly through partitioning can provide improvements over min-cut. Huang and Kahng showed that net weighting and quadrisection can be used to minimize a wide range of objectives such as minimal spanning tree cost [68]. Their technique consists of computing vectors of weights for each net (called net-vectors) and using these weights in quadrisection [68]. Although this technique can represent a wide range of cost functions to minimize, it requires the discretization of pin locations into the centers of bins and requires that sixteen weights must be calculated per net for partitioning [68].

The authors of [123] introduce a new terminal propagation technique in their placer THETO that allows the partitioner to better map net-cut to HPWL. The terminal propagation in THETO differs from traditional terminal propagation in that each original net may be represented by one or two nets in the partitioned netlist, depending on the configuration of the net's terminals. Two special cases — nets with no terminals and inessential nets — are treated the same as in traditional terminal propagation. Five other cases are analyzed in [123], based on the configuration of terminals relative to the centers of the child bins, and proper weight computation is described (one case requires two nets). This way weighted net-cut better represents the "HPWL degradation" seen after partitioning.

Empirically, this terminal propagation and net weighting are shown to reduce HPWL.

This technique is simplified in [37] and reduced to the calculation of three wirelengths per net per partitioning instance (w_1, w_2 and w_{12}) which completely determine the connectivity and costs of all nets in the derived partitioning hypergraph. While this formulation is more compact than that in [123], it is also more general. For each net in each partitioning instance, one must calculate the cost of all nodes on the net being placed in partition 1 (w_1), the cost of all nodes on the net being placed in partition 2 (w_2), and the cost of all nodes on the net being split between partitions 1 and 2 (w_{12}). Up to two nets can be created in the partitioning instance, one with weight $|w_1 - w_2|$ and the other with weight $w_{12} - max(w_1, w_2)$. The only assumption made in [37] is that $w_{12} \geq max(w_1, w_2)$. With these costs and the corresponding connectivity of the derived hypergraph, minimizing weighted net-cut directly corresponds to minimizing HPWL.

2.3 Mixed-size Placement

Mixed-size placement, the placement of large macros in addition to standard cells, has become a relevant challenge in physical design and is poised to dominate physical design in the near future as the industry moves from traditional "sea of cells" ICs to "sea of hard macros" SoCs [140]. To keep up with this shift in physical design, several techniques for partitioning based mixed-size placement have been proposed and are described in this section. These techniques include "floorplacement," fast look-ahead block-packing in PATOMA, and fractional cut.

2.3.1 Floorplacement

ALGORITHM 2.2: Min-cut floorplacement

```
    ▷ Input: queue of placement bins Q, netlist to place
    ▷ Output: placements of all the movable objects in netlist
 1  while (EMPTY(Q) = FALSE)
 2      do bin ← DEQUEUE(Q)
 3         if (BINHASLARGEMACRO(bin) = TRUE
            or GETNUMBEROFMACROS(bin) ≥ MACROTHRESHOLD
            or ISBINMERGED(bin) = TRUE)
 4           then CLUSTERSTDCELLSINTOSOFTMACROS(bin)
 5                success ← CALLFIXEDOUTLINEFLOORPLANNER(bin)
 6                if (success = TRUE)
 7                  then FIXMACROLOCATIONS(bin)
 8                       REMOVESITESBELOWMACROS(bin)
 9                  else newBin ← MERGEBINWITHPARTITIONINGSIBLING(bin)
10                       MARKBINASMERGED(newBin)
11                       ENQUEUE(Q, newBin)
12         else if (DETERMINEBINSIZE(bin) = SMALL)
13                then CALLENDCASEPLACER(bin)
14                else direction ← CHOOSECUTLINEDIRECTION(bin)
15                     cutline ← CHOOSECUTLINEPOSITION(bin, direction)
16                     graph ← BUILDPARTITIONINGGRAPH(bin, cutline, netlist)
17                     childBins ← CALLPARTITIONER(bin, cutline, graph)
18                     ENQUEUE(Q, childBins)
```

Figure 2.3: Min-cut floorplacement. Steps 3-11 differ from min-cut placement [115].

From an optimization point of view, floorplanning and placement are very similar problems – both seek non-overlapping placements to minimize wirelength. They are mostly distinguished by scale and the need to account for shapes in floorplanning, which calls for different optimization techniques. Netlist partitioning is often used in placement algorithms, where geometric shapes of partitions can be adjusted. This considerably blurs the separation between partitioning, placement and floorplanning, raising the possibility that these three steps can be performed by one CAD tool. We develop such a tool and term the unified layout optimization *floorplacement* following Steve Teig's keynote speech at ISPD 2002 [115].

Min-cut placers scale well in terms of runtime and wirelength minimization, but cannot produce non-overlapping placements of modules with a wide variety of sizes. On the other hand, annealing-based floorplanners can handle vastly different module shapes and sizes, but only for relatively few (100-200) modules at a time. Otherwise, either solutions will be poor or optimization will take too long to be practical. The loose integration of fixed-outline floorplanning and standard-cell placement proposed in [4] suffers from a similar drawback because its single top-level floorplanning step may have to operate on numerous modules. Bottom-up clustering can improve the scalability of annealing, but not sufficiently to make it competitive with other approaches. The work in [115] applies min-cut placement as much as possible and delays explicit floorplanning until it becomes necessary. In particular, since min-cut placement generates a slicing floorplan, it is viewed as an implicit floorplanning step, reserving explicit floorplanning for "local" non-slicing block packing.

Placement starts with a single placement bin representing the entire layout region with all the placeable objects initialized at the center of the bin. Using min-cut partitioning, the bin is split into two bins of similar sizes, and during this process the cut-line is adjusted according to actual partition sizes. Applying this technique recursively to bins (with terminal propagation) produces a series of gradually refined slicing floorplans of the entire layout region. In very small bins, all cells can be placed by a branch-and-bound end-case placer [23]. However, this scheme breaks down on modules that are larger than their bins. When such a module appears in a bin, recursive bisection cannot continue, or else will likely produce a placement with overlapping modules. Indeed, the work in [86] continues

bisection and resolves resulting overlaps later. In this technique, one switches from recursive bisection to "local" floorplanning where the fixed outline is determined by the bin. This is done for two main reasons: (i) to preserve wirelength [20], congestion [16] and delay [77] estimates that may have been performed early during top-down placement, and (ii) avoid legalizing a placement with overlapping macros.

While deferring to fixed-outline floorplanning is a natural step, successful fixed-outline floorplanners have appeared only recently [3]. Additionally, the floorplanner may fail to pack all modules within the bin without overlaps. As with any constraint-satisfaction problem, this can be for two reasons: either (i) the instance is unsatisfiable, or (ii) the solver is unable to find any of existing solutions. In this case, the technique undoes the previous partitioning step and merges the failed bin with its sibling bin, whether the sibling has been processed or not, then discards the two bins. The merged bin includes all modules contained in the two smaller bins, and its rectangular outline is the union of the two rectangular outlines. This bin is floorplanned, and in case of failure can be merged with its sibling. The process is summarized in Figure 2.3 and an example is depicted in Figure 2.4.

It is typically easier to satisfy the outline of a merged bin because circuit modules become relatively smaller. However, Simulated Annealing takes longer on larger bins and is less successful in minimizing wirelength. Therefore, it is important to floorplan at just the right time, and the algorithm determines this point by backtracking. Backtracking does incur some overhead in failed floorplan runs, but this overhead is tolerable because merged bins take considerably longer to floorplan. Furthermore, this overhead can be moderated somewhat by careful prediction.

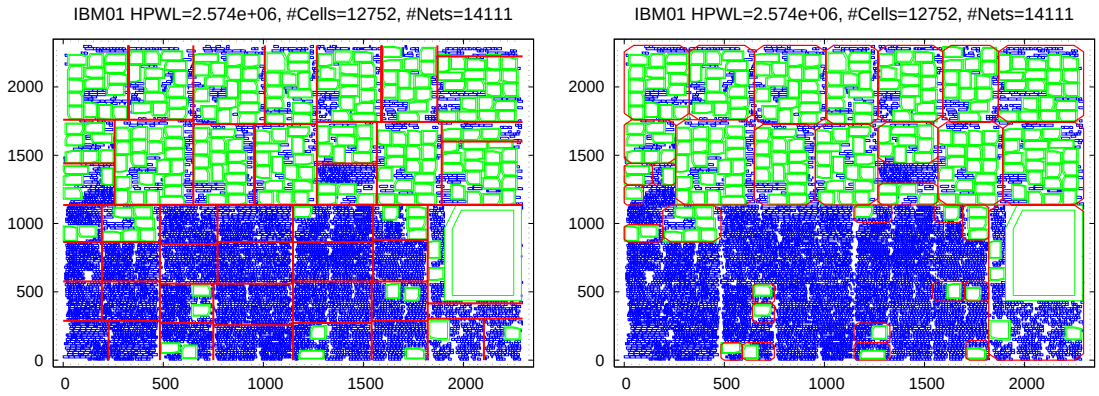


Figure 2.4: Progress of mixed-size floorplacement on the IBM01 benchmark from `IBM-MSwPins`. The picture on the left shows how the cut lines are chosen during the first six layers of min-cut bisection. On the right is the same placement but with the floorplanning instances highlighted by "rounded" rectangles. Floorplanning failures can be detected by observing nested rectangles [115].

For a given bin, a floorplanning instance is constructed as follows. All connections between modules in the bin and other modules are propagated to *fixed terminals* at the periphery of the bin. As the bin may contain numerous standard cells, the number of movable objects is reduced by conglomerating standard cells into soft placeable blocks. This is accomplished by a simple bottom-up connectivity-based clustering [84]. The existing large modules in the bin are usually kept out of this clustering. To further simplify floorplanning, soft blocks consisting of standard cells are artificially downsized, as in [5]. The clustered netlist is then passed to the fixed-outline floorplanner Parquet [3], which sizes soft blocks and optimizes block orientations. After suitable locations are found, the locations of all large modules are returned to the top-down placer and are considered fixed. The rows below those modules are fractured and their sites are removed, i.e., the modules are treated as fixed obstacles. At this point, min-cut placement resumes with a bin that has no large modules in it, but has somewhat non-uniform row structure. When min-cut placement is finished, large modules do not overlap by construction, but small cells some-

times overlap (typically below 0.01% by area). Those overlaps are quickly detected and removed with local changes.

Since the floorplacer includes a state-of-the-art floorplanner, it can natively handle pure block-based designs. Unlike most algorithms designed for mixed-size placement, it can pack blocks into a tight outline, optimize block orientations and tune aspect ratios of soft blocks. When the number of blocks is very small, the algorithm applies floorplanning quickly. However, when given a larger design, it may start with partitioning and then call fixed-outline floorplanning for separate bins. As recursive bisection scales well and is more successful at minimizing wirelength than annealing-based floorplanning, the proposed approach is scalable and effective at minimizing wirelength.

Empirical boundary between placement and floorplanning. By identifying the characteristics of placement bins for which the algorithm calls floorplanning, one can tabulate the empirical boundary between placement and floorplanning. Formulating such *ad hoc* thresholds in terms of dimensions of the largest module in the bin, etc., allows one to avoid unnecessary backtracking and decrease the overhead of floorplanning calls that fail to satisfy the fixed outline constraint because they are issued too late. In practice, issuing floorplanning calls too early (i.e., on larger bins) increases final wirelength and sometimes runtime. To improve wirelength, the *ad hoc* tests for large modules in bins (that trigger floorplanning) are deliberately conservative.

These conditions were derived by closely monitoring the legality of floorplanning and min-cut placement solutions. When a partitioned bin yields an illegal placement solution it is clear that the bin should have been floorplanned and a condition should be derived.

When a call to floorplanning fails to satisfy the fixed outline constraint the placer has to backtrack. To avoid paying this penalty, a condition should be derived to allow for floorplanning the parent bin and prevent the failure. A sample set of floorplanning conditions is shown in [115, Table III].

Conditions are refined to prevent floorplanning failure by visual inspection of a plot of the resulting parent bin and formulating a condition describing its composition. An example of such a plot is shown in Figure 2.4. Floorplanned bins are outlined with rounded rectangles. Nested rectangles indicate a failed floorplan run, followed by backtracking and floorplanning of the larger parent bin. In our experience, these tests are sufficient to ensure that at most one level of backtracking is required to prevent large module overlaps.

2.3.2 PATOMA and PolarBear

PATOMA 1.0 [49] pioneered a top-down floorplanning framework that utilizes fast block-packing algorithms (ROB or ZDS [48]) and hypergraph partitioning with hMETIS [84]. This approach is fast and scalable, and provides good solutions for many input configurations. Fast block-packing is used in PATOMA to guarantee that a legal packing solution exists, at which point the burden of wirelength minimization is shifted to the hypergraph partitioner. This idea is applied recursively to each of the newly-created partitions. In end-cases, when a partitioning step leads to unsatisfiable block-packing, the quality of the result is determined by the quality of its fast block-packing algorithms. In end-cases, when partitioning cannot be used because it creates unsatisfiable instances of block-packing, block locations are determined by fast block-packing heuristics. The placer PolarBear [50] integrates algorithms from PATOMA to increase the robustness of a top-

down min-cut placement flow. The floorplanner IMF [37] utilizes top-down partitioning, but allows overlaps in the initial top-down partitioning phase. A bottom-up merging and refinement phase fixes overlaps and further optimizes solution quality.

2.3.3 Fractional Cut for Mixed-size Placement

The work in [86] advocates a two-stage approach to mixed-size placement. First, the min-cut placer FengShui [6] generates an initial placement for the mixed-size netlist without trying to prevent all overlaps between modules. The placer only tracks the global distribution of area during partitioning and uses the *fractional cut* technique (see Section 2.2.2), which further relaxes book-keeping by not requiring placement bins to align to cell rows. While giving min-cut partitioners more freedom, these relaxations prevent cells from being placed in rows easily and require additional repair during detailed placement. This may particularly complicate the optimization of module orientations, not considered in [86] (relevant benchmarks use only square blocks with all pins placed in the centers).

The second stage consists of removing overlaps by a fast legalizer designed to handle large modules along with standard cells. The legalizer is essentially greedy and attempts to shift all modules towards the left edge of the chip (or to the right edge, if that produces better results). In our experience, the implementation reported in [86] leads to horizontal stacking of modules and sometimes yields out-of-core placements, especially when several very large modules are present (the benchmarks used in [86] contain numerous modules of medium size). See Figure 10 in [115] and Figure 6 in [107] for examples of this behavior. Another concern about packed placements is the harmful effect of such a strategy on routability, explicitly shown in [143]. Overall, the work in [86] demonstrates very good

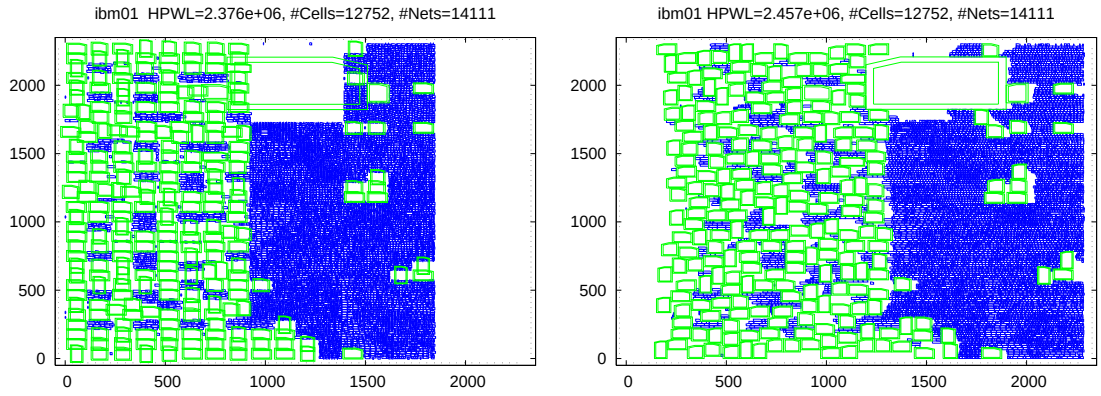


Figure 2.5: A placement of the IBM01 benchmark from `IBM-MSwPins` by FengShui before (left) and after (right) legalization and detailed placement.

legal placements for common benchmarks, but questions remain about the robustness and generality of the proposed approach to mixed-size placement. Example FengShui placements before and after legalization are shown in Figure 2.5.

2.3.4 Mixed-size Placement in Dragon2006

The traditional flow of the Dragon tool [144] does not take macros into consideration during placement. To account for macros, partitioning, bin-based annealing and legalization must be modified. In addition, Dragon2006 makes two passes on a design with obstacles; the first pass finds locations for macros and the second treats macros as fixed obstacles [130] (similar to [4]).

In the first pass, partitioning is modified to handle large movable macros. The traditional Dragon flow alternates cut directions at each layer and chooses the cut-line to split a bin exactly in half in order to maintain a regular grid structure. In the presence of large macros, the requirement of a regular bin structure is relaxed. The cut-line of the bin is shifted to allow the largest macro to fit into a child bin after partitioning. If macros can only fit in one bin, they are pre-assigned to the child bin in which they can fit and not

involved in partitioning [129, 130].

Bin-based Simulated Annealing after partitioning is also modified as bins may not all have the same dimensions. Horizontal swaps between adjacent bins are only allowed if they are of the same height. Similarly, vertical swaps between adjacent bins are only allowed if they are of the same width. Diagonal bin swaps are only legal if the bins have the same height and width. After all bins have a threshold of cells or fewer, partitioning stops and macro locations are legalized. Once legal, macros are considered fixed and partitioning begins again at the top level to place the standard cells of the design [129, 130].

2.3.5 Solving Difficult Instances of Floorplacement

Floorplacement (see Section 2.3.1) appears promising for SoC layout because of its high capacity and the ability to pack blocks. However, as experiments in [107] demonstrate, existing tools for floorplacement are fragile — on many instances they fail, or produce remarkably poor placements.

To improve the performance of min-cut placement on mixed-size instances, the authors of [107] propose three synergistic techniques for floorplacement that in particular succeed on hard instances: (i) selective floorplanning with macro clustering, (ii) improved obstacle evasion for B*-tree, and (iii) ad hoc look-ahead in top-down floorplacement. Obstacle evasion is especially important for top-down floorplacement, even for designs that initially have no obstacles. The techniques are called SCAMPI, an acronym for *SCalable Advanced Macro Placement Improvements*. Empirically, SCAMPI shows significant improvements in floorplacement success rate (68% improvement as compared to the floorplacement technique presented in Section 2.3.1) and HPWL (3.5% reduction compared to floorplacement

in Section 2.3.1).

Traditional placement techniques such as top-down and analytical frameworks, bottom-up clustering and iterative cell-spreading, scale well in terms of runtime and interconnect optimization *when all modules are small*. However, handling a wide variety of module sizes with these techniques seems considerably more difficult [107]. On the other hand, simulated annealing has a good track record in handling heterogeneous module configurations, but can only be effectively applied to small problem sizes [107]. This dichotomy between large-scale placement techniques and annealing-based floorplanning necessitates a rethinking of existing floorplacement flows [107].

Selective floorplanning with macro clustering. In top-down correct-by-construction frameworks like Capo and PATOMA [49] (see Section 2.3.2), a key bottleneck is in ensuring ongoing progress — partitioning, floorplanning or end-case processing must succeed at any given step. Both frameworks experience problems when floorplanning is invoked too early to produce reasonable solutions — PATOMA resorts to solutions with very high wirelength, and Capo times out because it has nothing to resort to and runs the an annealer on too many modules. In order to scale better, the annealer clusters small standard cells into soft blocks before starting Simulated Annealing. When a solution is available, all hard blocks are considered placed and fixed — they are treated as obstacles when the remaining standard cells are placed. Compared to other multi-level frameworks, this one does not include refinement, which makes it relatively fast. Speed is achieved at the cost of not being able to cluster modules other than standard cells because the floorplanner does not produce locations for clustered modules. Unfortunately, this limitation significantly

ALGORITHM 2.3: SCAMPI Min-cut floorplacement

```
    ▷ Input: queue of placement bins Q, netlist to place
    ▷ Output: placements of all the movable objects in netlist
 1  while (EMPTY(Q) = FALSE)
 2      do bin ← DEQUEUE(Q)
 3         passedLookahead ← TRUE
 4         if (ISBINMERGED(bin) = FALSE)
 5           then passedLookahead ← CALLLOOKAHEADFLOORPLANNER(bin)
 6                if (passedLookahead = FALSE)
 7                  then newBin ← MERGEBINWITHPARTITIONINGSIBLING(bin)
 8                       MARKBINASMERGED(newBin)
 9                       ENQUEUE(Q, newBin)
10         if (passedLookahead = TRUE)
11           then if (BINHASLARGEMACRO(bin) = TRUE
                  or GETNUMBEROFMACROS(bin) ≥ MACROTHRESHOLD
                  or ISBINMERGED(bin) = TRUE)
12                  then MARKLARGEMACROSFORPLACEMENTAFTERFLOORPLANNING(bin)
13                       CLUSTERUNMARKEDMACROSINTOSOFTMACROS(bin)
14                       CLUSTERSTDCELLSINTOSOFTMACROS(bin)
15                       success ← CALLFIXEDOUTLINEFLOORPLANNER(bin)
16                       if (success = TRUE)
17                         then FIXMACROLOCATIONS(bin)
18                              REMOVESITESBELOWMACROS(bin)
19                         else newBin ← MERGEBINWITHPARTITIONINGSIBLING(bin)
20                              MARKBINASMERGED(newBin)
21                              ENQUEUE(Q, newBin)
22                  else if (DETERMINEBINSIZE(bin) = SMALL and
                         GETNUMBEROFMACROS(bin) ≥ MACROPERCENTTHRESHOLD ×
                         GETNUMBEROFMOVABLEOBJECTS(bin))
23                         then CALLSTDCELLANDMACROENDCASEPLACEMENT(bin)
24                         else if (DETERMINEBINSIZE(bin) = SMALL)
25                                 then CALLENDCASEPLACER(bin)
26                                 else direction ← CHOOSECUTLINEDIRECTION(bin)
27                                      cutline ← CHOOSECUTLINEPOSITION(bin, direction)
28                                      graph ← BUILDPARTITIONINGGRAPH(bin, cutline, netlist)
29                                      childBins ← CALLPARTITIONER(bin, cutline, graph)
30                                      ENQUEUE(Q, childBins)
```

Figure 2.6: SCAMPI floorplacement flow. Steps 3-10, 12-13, and 22-23 differ from traditional floorplacement [107].

restricts scalability of designs with many macros [107].

The proposed technique of *selective floorplanning with macro clustering* allows clustering of blocks before annealing, and does not require additional refinement or cluster-packing steps (which are among the obvious facilitators) — instead certain existing steps in floorplacement are skipped. This improvement is based on two observations: (i) blocks

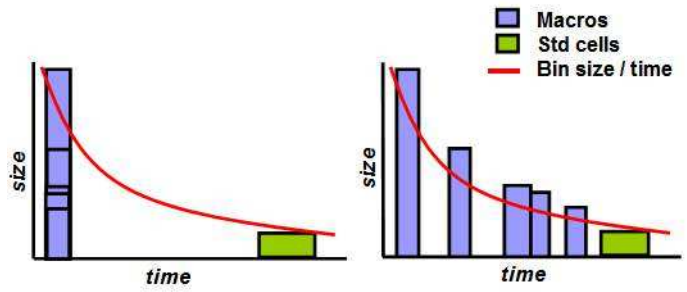


Figure 2.7: The plot on the left illustrates traditional floorplacement. Whenever a floorplanning threshold is reached, all macros in the bin are designated for floorplanning. Then, the floorplacement flow continues down until detailed placement, where the standard cells will be placed. The plot on the right illustrates the SCAMPI flow. Macros are selectively placed at the appropriate levels of hierarchy [107].

that are much smaller than their bin can be treated like standard cells, (ii) the number of blocks that are large relative to the bin size is necessarily limited, e.g., there cannot be more than nine blocks with area in excess of 10% of a bin's area [107].

In selective floorplanning, each block is marked as small or large based on a size threshold. Standard cells and small blocks can be clustered, except that clusters containing hard blocks have additional restrictions on their aspect ratios. After successful annealing, only the large blocks are placed, fixed and considered obstacles. Normal top-down partitioning resumes, and each remaining block will qualify as large at some point later. This way, specific locations are determined when the right level of detail is considered. If floorplanning fails during hierarchical placement, the failed bin is merged with its sibling and the merged bin is floorplanned (see Figure 2.6). The blocks marked as large in the merged bin include those that exceed the size threshold and also those marked as large in the failed bin (since the failure suggests that those blocks were difficult to pack). After the largest macros are placed, the flow resumes [107].

The proposed technique limits the size of floorplanning instances given to the annealer *by a constant* (in our case 200 modules) and does not require much extra work. However, it introduces an unexpected complexity. The floorplacement framework does not handle fixed obstacles in the core region, and none of the public benchmarks have them. When Capo fixes blocks in a particular bin, it fixes *all of them* and never needs to floorplan around obstacles. Another complication due to newly introduced fixed obstacles is in cut-line selection. Reliable obstacle-evasion and intelligent cut-line selection may be required by practical designs, even without selective floorplanning (e.g., to handle pre-diffused memories, built-in multipliers in FPGAs, etc). Therefore they are viewed as independent but synergistic techniques [107].

Obstacle evasion in floorplanning: B*-tree enhancement. When satisfying area constraints is difficult, it is very important to increase the priority of area optimization so as to achieve legality [36]. Because of this, the authors of [107] select the B*-tree [32] floorplan representation for its amenability to packed configurations and add obstacle evasion into B*-tree evaluation.

Ad-hoc look-ahead floorplanning. The sum of block areas may significantly underestimate the area required for large blocks. Better estimates are required to improve the robustness of floorplacement and look-ahead area-driven floorplanning appears as a viable approach [107].

SCAMPI performs look-ahead floorplanning to validate solutions produced by the hypergraph partitioner, and check that a resulting partition is packable, within a certain tolerance for failure. Look-ahead floorplanning must be fast, so that the amortized runtime

overhead of the look-ahead calls is less than the total time saved from discovering bad partitioning solutions. Therefore look-ahead floorplanning is performed with blocks whose area is larger than 10% of the total module area in the bin, and soft blocks containing remaining modules. For speed, the floorplanner is configured to perform area-only packing, and the placer is configured to only perform look-ahead floorplanning on bins with large blocks. Dealing with only the largest blocks is sufficient because floorplanning failures are most often caused by such blocks [107].

Selective floorplanning for multi-million gate designs. One case that is not considered by either the original floorplacement techniques [115] or those introduced in the original SCAMPI flow [107] is where there are an extreme number of movable modules and an extreme ratio between the largest and smallest macro. An example of this is the `newblue1` benchmark from the ISPD'06 placement contest suite [102]. The `newblue1` benchmark contains 64 macros and 330073 standard cells. As we show below, such a configuration is problematic for floorplacement.

Recall that a floorplacer utilizes a floorplanner to place macros. As the floorplanner uses Simulated Annealing to pack blocks, clustering is performed on the netlist to improve scalability. However, a very large number of small modules may stress clustering algorithms, which, in the absence of refinement, may undermine overall solution quality.[1]

In the original SCAMPI flow, the largest block in the `newblue1` benchmark was designated for floorplanning by Parquet at the top level. Parquet precedes annealing with clustering to reduce the size of the netlist. However, given the large number of small modules, Parquet's simple-minded clustering algorithm took 16% of total floorplacement runtime,

[1] Refinement algorithms would need to operate on very large netlists and may require long runtimes.

whereas annealing took only 4% of floorplacement runtime (with the remaining 80% consumed in standard cell placement). Additionally, even if clustering were more scalable, clustering such a large number of small macros into large, soft macros can lead to unnatural or unrepresentative netlists. In the original SCAMPI flow, the clusters formed by the standard cells in `newblue1` became large enough to artificially constrain the movement of the large macro during floorplanning. This is mainly a limitation of Simulated Annealing as it becomes impractical in solution quality and runtime for over 100 modules.

To counteract this undesirable effect, the authors of [118] propose the following method. Whenever a bin is designated for floorplanning and the largest real module is smaller in area than the largest soft macro built from clustering (this area can be estimated without actually performing clustering), do not use Simulated Annealing. Instead, use a simple analytical tool to minimize quadratic wirelength to determine reasonable locations for the large macros. It has been shown that analytical techniques are good at finding general areas where objects should be placed [10], so this is a reasonable and efficient solution for placing a large macro or macros in this situation. This technique may also be useful in regions with large amounts of whitespace as block-packing often overlooks good solutions in such situations. Objectives other than HPWL, such as routing congestion and timing, are also important, and any analytical placer used in this context should place macros with respect to the most relevant objective(s). The key observation is that placing such macros early is helpful.

When there is only one large macro to be placed, the solution of the analytical tool is used and the macro is fixed in its desired location. To place a small number of large

Table 2.1: Comparison of Capo's performance at the ISPD 2006 Placement Contest. "Overflow" represents the HPWL penalty for not effectively enforcing density constraints on the benchmarks. Using the SCAMPI improvements, Capo's HPWL is reduced by 7% overall.

Benchmark	Capo at ISPD 2006		Capo + improved SCAMPI		
	HPWL (e8)	Overflow%	HPWL (e8)	Overflow%	HPWL Ratio
adaptec5	4.92	0.62	4.88	0.42	0.99
newblue1	0.98	0.13	0.81	0.12	0.83
newblue2	3.09	0.29	2.67	0.21	0.86
newblue3	3.61	0.01	3.35	0.01	0.93
newblue4	3.58	1.15	3.51	0.83	0.98
newblue5	6.57	0.33	6.41	0.26	0.98
newblue6	6.68	0.05	6.53	0.05	0.98
newblue7	15.18	0.02	13.64	0.01	0.90
Average					0.93

macros with this method, we again compute macro locations with the analytical tool, but must legalize the macro locations to maintain the correct-by-construction paradigm of floorplacement. Overlaps can be legalized in several ways. One way is to use a greedy macro legalization technique such as the macro legalizer described in [115, Section 3.3]. Another method for removing macro overlap is the constraint-based floorplan repair algorithm FLOORIST [100]. Following legalization, one can shift the macros, making sure they remain in the core area, so that their center of mass coincides with their center of mass before legalization in keeping with the spirit of the analytical placement. This technique contributed to HPWL improvement over the ISPD 2006 Placement Contest results of Capo by 17% on newblue1 as shown in Table 2.1, with an overall improvement in the contest score on the ISPD 2006 benchmark suite by 10%, moving Capo three positions higher.

Temporary macro deflation. Low-whitespace conditions in block-packing instances formed during floorplacement can worsen solution quality significantly. In such cases, the

block-packing engine focuses mainly on finding legal solutions rather than those that have good wirelength. In addition, a legal solution may not be found, which leads to backtracking and increased runtime as well. To improve the solution quality of block-packing instances created during floorplacement, we prevent these low-whitespace conditions.

To account for standard cells in the floorplacement framework, they are clustered into soft blocks for instances of block-packing [115]. To improve the likelihood of finding a legal fixed-outline solution, these soft blocks representing standard cells are reduced in size [115]. We propose extending this deflation to include hard blocks in addition to soft blocks. When a block-packing instance is formed, we adjust the sizes of hard blocks to maintain a minimum amount of whitespace. All blocks in the instance are sized in the same way and aspect ratios are maintained. The resized instance, made easier by the addition of whitespace, is placed using Simulated Annealing as normal.

Resizing the hard blocks in this way has the positive effect of making fixed-outline block-packing easier, which allows the block-packing engine to focus on HPWL minimization rather than mere legality in cases where whitespace is limited, but removes the correct-by-construction property upon which floorplacement is built. To alleviate this problem, we apply legalization to macros after packing using the fast and robust constraint-based floorplan repair algorithm FLOORIST [100] after each layer of placement where block-packing took place. FLOORIST moves macros minimally when repairing overlaps, so reduced HPWL found in easier block-packing instances is preserved.

Empirically we find that the overhead of running FLOORIST for legalization is mitigated by the fact that block-packing is easier and therefore faster. In terms of solution

quality, we find that temporary macro deflation reduces HPWL by 2-3%.

2.4 State-of-the-art Min-cut Placers

In this section, we outline partitioning-based placement techniques used by cutting-edge placers. For each placer, we describe its overall flow, how this differs from the basic min-cut flow (Figure 2.1), and how it handles challenges in placement such as fixed obstacles and mixed-size instances. In particular we describe the techniques used by Dragon [129, 130, 143], FengShui [6, 86], NTUPlace2 [74] and Capo [107, 115–117, 119, 120].

2.4.1 Dragon

Dragon combines min-cut bisection with Simulated Annealing for placement [130]. In its most basic flow, Dragon2006, the most recent version of Dragon, utilizes recursive bisection with the hMETIS partitioner [84]. Each bin is partitioned multiple times with a feedback mechanism to allow for more accurate terminal propagation (see Section 2.2.1 for more details on placement feedback). Partitioning is followed by Simulated Annealing on the placement bins where whole bins are swapped with one another to improve HPWL [129, 130]. After a number of layers of interleaved partitioning and Simulated Annealing, each bin contains only a few cells and the partitioning phase terminates. Next, bins are aligned to row structures and cell-based Simulated Annealing is performed wherein cells are swapped between bins to improve HPWL [129, 130]. Lastly, cell overlaps are removed and local detailed placement improvements are made. Mixed-size placement is supported as described in Section 2.3.4.

2.4.2 FengShui

FengShui [6, 86] is a recursive bisection min-cut placer that uses the hMETIS partitioner [84]. FengShui implements the fractional cut technique (see Section 2.2.2) and packs its placements to either side of the placement region which has a serious affect on the routability of its placements [116]. FengShui also supports mixed-size placement (see Section 2.3.3).

2.4.3 NTUPlace2

NTUPlace2 [74] is a hybrid placer that uses both min-cut partitioning and analytical techniques for standard-cell and mixed-size designs. NTUPlace2 uses repartitioning (see Section 2.2.1), cut-line shifting (see Section 2.1.3) and weighted net-cut (see Section 2.2.4) [74].

NTUPlace2 uses analytical techniques to aid partitioning which are different from those in ACG (see Section 2.2.3). Before partitioning calls to the hMETIS partitioner [84], objects in a placement bin are first placed by an analytical technique to reduce quadratic wirelength [74]. Those objects which are placed far from the proposed cut-line are considered fixed in their current locations for the partitioning process. This technique helps to make terminal propagation more exact and with the weighted net-cut technique has resulted in very good solution quality [74].

To handle mixed-size placement, macro locations are legalized at each layer. Macros become fixed at different layers of placement according to their size relative to placement bin size. Thus larger macros are placed earlier in placement [74]. Macros are legalized using a linear programming technique that attempts to minimize the movement of macros

during legalization [74].

2.4.4 Capo

Capo [107, 115–117, 119, 120] is a min-cut floorplacer. Capo implements the floorplacement flow (Section 2.3.1) and further improved by SCAMPI (Section 2.3.5) rather than the traditional min-cut flow. As a result, Capo implicitly handles mixed-size placement and fixed obstacles in the placement area. Capo can use either MLPart [22] or hMETIS [84] for hypergraph partitioning. Whitespace allocation in Capo is done per placement bin: either *uniform* (see Section 2.1.4), *minimum local* or *safe* whitespace allocation is chosen based on the bin's whitespace and user-configurable options. These whitespace allocation options are described in more detail in Chapter IV. To improve the quality of results, Capo also implements repartitioning (see Section 2.2.1), placement feedback (see Section 2.2.1), weighted net-cut (see Section 2.2.4) and routability-driven whitespace allocation (Chapter V). Capo has also been extended to perform incremental placement (Chapter VI) and to integrate global routing techniques (Chapter IX).

CHAPTER III

State of the Art in Global Routing

VLSI routing is an active area of research and development, as evidenced by a growing body of literature [7, 40, 64, 109, 110], recent collaboration between Cadence and IBM on routing technology [97], as well as the ISPD 2007 and 2008 Global Routing Contests organized by IBM Austin Research Laboratory [71]. Current efforts in routing are motivated by challenges present at the nanometer scale including: (i) very large wiring databases that require lean data structures and extremely efficient algorithms, (ii) sophisticated design rules that must be abstracted away during initial routing passes, (iii) relatively unreliable vias whose resistance may vary by up to 30 times [121], which requires via doubling [91, 94] and motivates additional effort to minimize via counts, (iv) signal integrity constraints and the dramatic impact of lateral capacitance on interconnect delay, which lead to wire-density constraints, and (v) considerations of chemical mechanical polishing (CMP) that also lead to density constraints [41].

3.1 Global and Detailed Routing

Routing plays a key role in VLSI physical design as it determines the specific shape and layout of interconnect, impacting performance, power and manufacturability. Routing

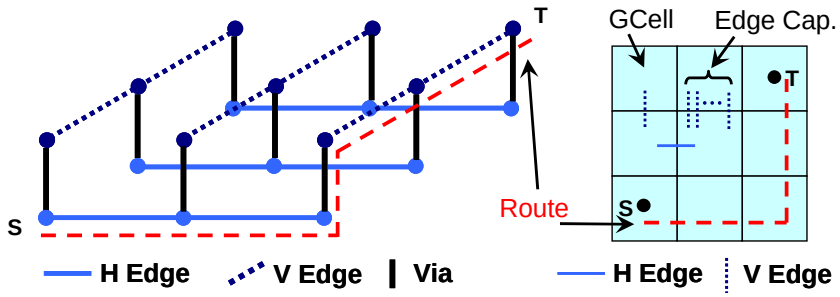


Figure 3.1: Pictorial representations of the global routing grid. The image on the right shows how the layout is abstracted into a regular grid of GCells. GCells are represented by vertices, with adjacent vertices connected by graph edges. Capacities on edges that join GCells can be defined as the number of routing tracks that cross GCell boundaries. The image at the left shows horizontal, vertical and via connections on different layers.

is traditionally divided into the two steps of global and detailed routing.

At the beginning of global routing, the design is abstracted into a regular three-dimensional *routing grid* with one layer for each layer of metal in the design. Grid cells, or *GCells*, on the same layer are connected with *routing edges*. An example routing grid with one layer is shown in Figure 3.1. Each GCell is connected to at most four neighbors on the same layer in the four cardinal directions. GCells on different layers are connected through *via* edges. A GCell may be connected to at most two neighbors, one layer above and one layer below, by vias. Each routing edge is assigned a capacity which represents the amount of metal, be it from wires or routing blockages, which is allowed to be routed from one GCell to its neighbor. Since different metal layers may use distinct wire pitches, routing capacities at each layer may differ to reflect this. We define the *congestion* of a routing edge as the ratio of the metal assigned to it by the router and blockages to its capacity. Typically one says that a routing edge is *congested* if more metal is assigned to it than its capacity allows. Typically the global router chooses the dimensions of GCells based on the size of the design then induces routing edge capacities. A global routing solution is legal if all

nets are connected and all capacity constraints are satisfied.

During global routing, the pins of each net are binned into GCells, and the global router is tasked with connecting all nets with metal through routing edges and via edges while simultaneously respecting routing edge capacity constraints. Nets which are wholly subsumed in a GCell are ignored by a global router in the academic literature and ISPD routing contests, but are generally accounted for by industrial tools. Detailed routing begins with a global routing solution, which may not necessarily satisfy all capacity constraints, and assigns wires for all nets to actual routing tracks, using the global routing solution as a guide. An industrial detailed router can handle a small number of global routing capacity constraint violations and produce a routing solution with all wires electrically connected (having no *opens*) and no metal from different nets electrically connected (no *shorts*).

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

In our experience with Cadence WarpRoute, three quarters of total runtime are spent in detailed routing, but the quality of global routes profoundly affects the runtime and success of detailed routing. A recent proposal [109] suggests invoking a fast global router during global and detailed placement, so as to mitigate wiring congestion early. This application is particularly attractive for sub-130nm technology nodes where lateral capacitance of wires is a major contributor to interconnect delay. In this context, accurate timing analysis

requires information about regions through which a given net passes as well as wire density in these regions [127].

3.2 Popular Global Routing Techniques

In this section we outline several popular techniques in the literature for global routing. These techniques include the routing of two-pin connections with shortest path search and patterns, various methods for handling nets with three or more pins, rip-up-and-re-route, congestion amplification, negotiated-congestion routing and multi-level algorithms.

3.2.1 Maze Routing

Maze routing connects pairs of terminals on the routing grid using standard search techniques such BFS and Dijkstra's algorithm [54]. More than 50% of nets in modern designs connect only two pins. BFS can find the shortest path between a source location and a target location, if one exists, but cannot handle routing segments with non-trivial weights. Dijkstra's algorithm can handle non-negative costs of routing segments, but is at least several times slower than BFS. A*-search is a minor modification to Dijkstra's algorithm that significantly improves speed during two-dimensional and three-dimensional routing [67]. In A*-search, a lower bound of the distance to the target is added to node priority in Dijkstra's algorithm. Straight-line distance is commonly used as a lower bound.

3.2.2 Pattern Routing

Pattern routing [85] simplifies the routing process by restricting the path that a net can take to one of a handful of pre-determined shapes. For example, L-shape routing seeks to implement each two-pin net with at most one bend. This technique is surprisingly useful

in ASIC routing and justified by via minimization. Empirical studies [141] show that in a fully-routed design a majority of all two-pin nets take on L-shapes. In global routing, where minor detours are abstracted away, L-shapes are even more prevalent. Two-bend routes are often called Z-shapes, but generic pattern-based routing can consider any finite number of routing topologies for each net, and selects one of them. It is particularly amenable to integer linear programming formulations [40], as described later in the section.

3.2.3 Handling Multi-pin Nets

Most global routing algorithms decompose nets with three or more pins into two-pin subnets at the beginning of global routing to ease maze routing. This decomposition has traditionally been done using Minimal Spanning Tree (MST) algorithms, but fast and extremely accurate Rectilinear Steiner Minimal Tree (RSMT) construction algorithms have become increasingly popular in the literature [40, 109, 110]. Four decompositions of a 5-pin net by Steiner trees and MSTs are shown in Figure 3.2.

The RSMT tool FLUTE [44] is used in both BoxRouter [40] and FastRoute [109, 110]. FLUTE uses look-up tables for nets with nine or fewer pins and quickly builds optimal trees for such nets [44]. For larger nets, a divide-and-conquer method is employed [44]. FastSteiner [78] is another RSMT algorithm that is more scalable than most RSMT algorithms. FastSteiner does not guarantee optimality, but frequently produces solutions with smaller total wirelength than FLUTE for nets with more than nine pins.

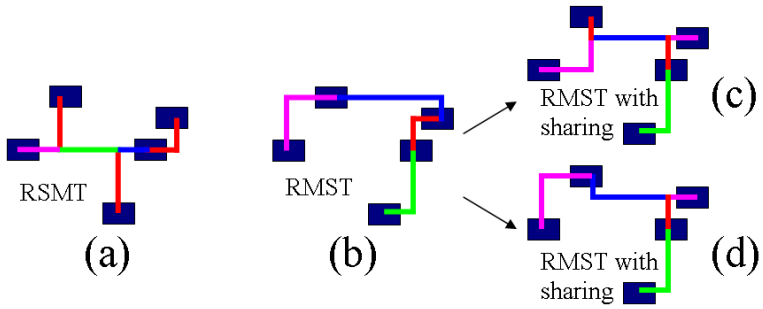


Figure 3.2: Decomposition of a 5-pin net by minimal Steiner tree (a), MST (b) and MSTs with sharing (c)&(d). The choice of (c) or (d) depends on congestion. The minimal Steiner tree (a) contains 5 flat subnets and 1 L-shaped subnet, whereas the shared MST (d) has 2 flat subnets and 3 L-shaped subnets which gives it greater flexibility.

3.2.4 Rip-up-and-re-route

Rip-up-and-re-route (RRR) takes an initial, usually illegal, routing solution and iterates greedy one-net-at-a-time routing passes for nets that compete for routing resources, but may change the ordering each time in hope to better reconcile these nets. In each iteration, nets that pass through congested regions are "ripped up" (all resources for the net are removed from the routing grid) and are rerouted with a maze router to use lesser congested regions. Major differences between various implementations [40, 55, 64, 98, 109, 110] include which nets are ripped up and rerouted at each iteration, the order in which to rip up nets and reroute them, if nets are allowed to be rerouted through areas that are already congested, and the costs associated with routing through a particular routing edge given its current congestion.

3.2.5 Congestion Amplification

Congestion amplification [64] was recently introduced as an improvement to pricing of routing resources during RRR. Many routers that employ RRR do not penalize nets for

going through uncongested regions, and then drastically increase cost once a routing edge is full. The authors of [64] propose to use a more gradual linear cost function for edges before they become full in order to spread wires from areas that are likely to become congested. In addition, when congestion estimates are calculated after each iteration of RRR, regions with high congestion have their estimates artificially increased (amplified) and regions with low congestion have their estimates decreased. This provides more incentive for maze routers to avoid highly congested regions at the cost of longer wirelength.

3.2.6 Negotiated-congestion Routing

Negotiated-congestion routing (NCR) [98] was introduced in the mid-1990s for global routing in FPGAs and is used in VPR (the dominant place-and-route tool for FPGAs) [14], but has not seen much use in the ASIC literature. NCR builds upon RRR by gradually making routing edges that are consistently congested more expensive, encouraging the maze router to choose alternative routes. The cost c_e of routing edge e

$$c_e = (b_e + h_e) \cdot p_e \tag{3.1}$$

is a function of the base cost (b_e), added cost reflecting congestion history (h_e), and penalty for current congestion (p_e) [98]. NCR seeks to minimize $\sum_e c_e$.

To begin negotiated-congestion routing, each net is routed using the smallest possible wirelength regardless of edge capacities. At the beginning of an RRR iteration, the historical cost h_e of all over-capacity routing edges is increased:

$$h_e^{k+1} = \begin{cases} h_e^k + h_{inc} & \text{if } e \text{ is overfull} \\ h_e^k & \text{otherwise} \end{cases} \tag{3.2}$$

where h_{inc} is a constant. The choice of h_{inc} affects convergence time and solution quality: higher values lead to faster convergence but higher routed wirelength. After cost adjustment, each net of the design is individually ripped up and rerouted by a maze router. The authors suggest that only nets passing through congested regions need to be rerouted. The ordering of nets during rip-up-and-re-route is the same for each iteration, but can be chosen arbitrarily, according to the authors of [98], because the gradual cost increase in congested areas removes ties that require sophisticated net ordering techniques in traditional RRR.

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

3.2.7 Multi-level Routing

Multi-level techniques for routing work similarly to those in partitioning [84] and placement [26]. The original routing problem is effectively made simpler through a series of coarsening stages where routing grid cells are combined and many nets become subsumed within a single cell. This adds a hierarchy to the routing formulation. This process is depicted in Figure 3.3. At the top of the hierarchy is the coarsest form of the routing problem which is small enough to be solved with sophisticated techniques that may not scale to large routing instances such as multi-commodity flow based techniques [7, 67], described in Section 3.2.8 below. Essential to the coarsening stage is the proper aggregation of routing resources so that routing solutions at higher levels closely resemble valid routing solutions at lower levels.

After the coarsest level of the hierarchy has been routed, iterative refinement of the

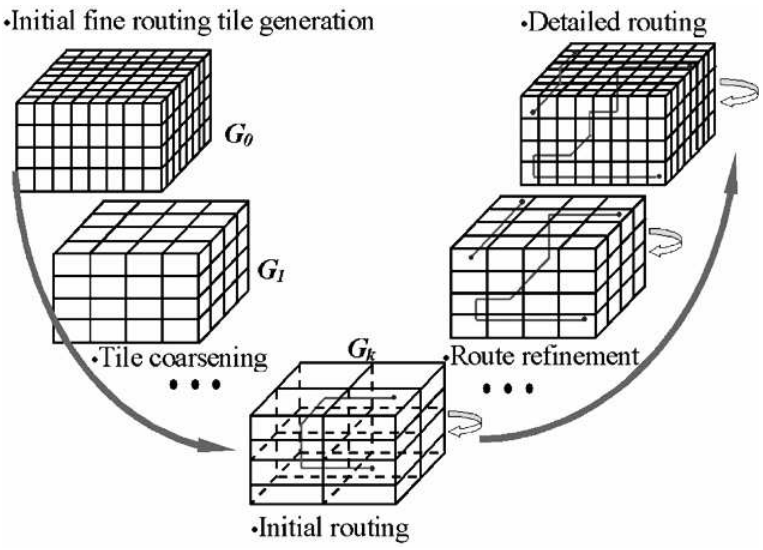


Figure 3.3: Multi-level routing progression; image from [53].

current routing solution begins. The problem is un-coarsened by one level and the current solution is adapted to the finer routing grid. This stage is nontrivial as nets can gain additional pins as the routing grid is refined and new nets that were previously subsumed by routing cells will become visible and need to be routed from scratch. This refinement process proceeds iteratively until the finest level of the hierarchy, the original routing problem, has been successfully routed. Multi-level routers in the literature generally have smaller runtimes than flat techniques and show higher completion rates [46, 53].

3.2.8 Combinatorial Optimization Techniques

Other sophisticated techniques for routing have been proposed, such as the use of multi-commodity flows (MCF) [7, 67] and integer linear programming (ILP) [40]. Both of these techniques attempt to route nets simultaneously in order to avoid the problems associated with net ordering.

ILP formulations for routing vary widely. Many require that multi-pin nets have been

maximize:	$\sum(x_{i1} + x_{i2})$	$\forall$ net i
such that:	$x_{i1}, x_{i2} \in \{0, 1\}$	$\forall$ net i
	$x_{i1} + x_{i2} \leq 1$	$\forall$ net i
	$\left(\sum_{\text{pattern } j \text{ of net } i \text{ uses edge } e} x_{ij}\right) \leq cap(e)$	$\forall$ routing edge e

Figure 3.4: BoxRouter ILP formulation for routing nets using only L shapes.

divided into a small number of two-pin topologies, but even more general encodings are possible. For each two-pin net, one must set up several constraints. One constraint is necessary per terminal asserting at least one routing segment be connected to the terminal for the net. A different type of constraint is used for each non-terminal GCell that makes sure exactly 0 or 2 routing segments attach to this GCell for the net. Constraints are also added per routing segment to maintain routing resource limits. The optimization objective of the ILP is to minimize the number of used routing segments. Solving this formulation will optimally solve the given routing problem if possible, but has its drawbacks including difficulty expressing non-linear delay models.

Another serious drawback is that the general ILP formulation requires a considerable number of variables and constraints, and does not scale well to large instances which are difficult to route. To make the technique more scalable, one must severely limit the number of ways that a two-pin net can be routed. This can be done by defining a small number of patterns per net and allowing the ILP solver to choose from among them. Indeed, BoxRouter restricts nets to only L shapes, which speeds ILP solving considerably, making it more practical [40]. As L shapes are optimal in wirelength, the BoxRouter formulation instead chooses to maximize the number of routed nets rather than minimize routing segment use. The BoxRouter ILP formulation is given in Figure 3.4.

Multi-commodity flow (MCF) techniques take a different approach to solving routing.

They begin with an ILP routing formulation and relax it into a linear programming (LP) formulation by changing the Boolean variables representing routing edge use into non-negative real-valued variables. An approximation algorithm which successively adjusts routing edge weights and builds new weighted Steiner trees per net at each iteration is used to solve the LP. BoxRouter has been compared to a recent MCF-based router [40] and found to be superior in speed and solution quality. Additionally, MCF techniques offer less flexibility in terms of objective functions and constraints than the RRR and NCR frameworks.

3.3 State-of-the-art Global Routers

In this section, we present the techniques used by leading global routers. For each router, we describe its overall flow and how it combines routing described in Section 3.2 above. In particular we describe the methods employed by FastRoute [109, 110], BoxRouter [40] and the multi-level router MARS [53].

3.3.1 FastRoute

FastRoute [109, 110] uses a simplified, more greedy form of RRR and finishes orders of magnitude faster than other routers. However, it was able to legally route only 6 of 16 benchmarks at the ISPD '07 contest [71], while other routers completed up to 12 benchmarks without violations.

FastRoute 1.0 [109] first uses FLUTE to decompose nets and estimate congestion in the design, then attempts to restructure Steiner trees to avoid congestion. FastRoute 2.0 [110] features the following modification of RRR. When a single subnet is ripped up, the net to

which the subnet belongs will be separated into two connected components. It becomes the maze router's job to connect the two components of the net in the least costly way. While this optimization allows the router to move Steiner points away from congested regions, it invalidates the point-to-point lower bound on which A*-search relies. Therefore, the slower Dijkstra's algorithm must be used instead.

3.3.2 BoxRouter

BoxRouter [40] avoids fine-grain net ordering in congested regions through the use integer linear programming (ILP) formulations. BoxRouter decomposes nets using Steiner trees produced with FLUTE but never re-examines their decomposition. Next it performs a pass of pattern routing that identifies the most congested rectangular region, where it formulates an ILP to route as many nets using L-shapes as possible. Remaining nets are routed by the maze router, using as few resources outside the region as possible. Next, the region is expanded, and an incremental ILP formulation is used. This cycle repeats until the entire layout is covered by the expanding region.

3.3.3 MARS

The Multilevel Advanced Routing System (MARS) [53] is a multi-level router (see Section 3.2.7) based on the techniques first presented in [46] with several important enhancements. The first is that MARS performs accurate resource reservation during the coarsening phase of multi-level routing. This takes into account those nets which are subsumed into the coarsened routing grid and removes resources for them. This results in more accurate resource counts at higher levels of the routing hierarchy which better represent the original routing problem. The second enhancement is that MARS divides

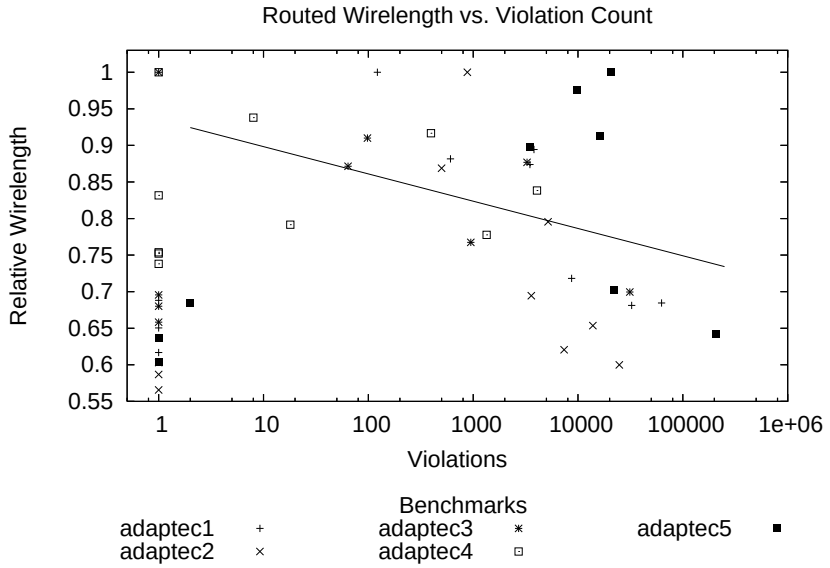


Figure 3.5: Routed wirelength versus violations for all competing routers on two-dimensional ISPD '07 benchmarks [71]. Note that violation counts are shown on a log-scale where 0 cannot be plotted, so completely legal solutions are depicted with exactly 1 violation. Relatively few solutions submitted to the contest were legal (35%), but they are generally a cut above the rest. Of the illegal solutions, as violations increase, routed wirelength decreases. To emphasize the trend, a linear least-squares fit of the data has been added for the illegal solutions.

multi-pin nets using congestion-driven Steiner trees. At each level of the routing hierarchy, each net is examined and new Steiner trees are built to divide multi-pin nets. First MSTs are built for each net using the routing grid and not purely based on HPWL. Next, the edges of the MST for a particular net are sorted based on length and maze search is performed to join the edge to any other part of the existing tree. The new attachment points become Steiner points, and the Steiner tree for the net is formed from all of the paths found during maze search. Lastly, MARS uses historical costs based on congestion, similar to those described in Section 3.2.6, to price routing edges during maze routing.

3.4 The ISPD '07 Routing Contest

The ISPD '07 routing contest challenged the research community by distributing 16 very large routing benchmarks derived from recent chip layouts. The contest had separate two-dimensional and three-dimensional divisions and a total of 9 academic teams competed. The top teams in the two-dimensional contest were 1) our router FGR which is detailed in Chapter VII, 2) MaizeRouter, 3) BoxRouter and 4) FastRoute. In the three-dimensional division the standings were as follows: 1) MaizeRouter, 2) BoxRouter, 3) FGR and 4) FastRoute.

Thanks to the wide participation in the contest and the public availability of the results, we observed an important trend which is illustrated in Figure 3.5 — routers that achieve low wirelength often suffer high violation counts, and routers that minimize violations often produce high wirelengths. Therefore, a key focus of our work is on adequate pricing of routing resources to balance interconnect length and congestion in multi-million gate designs, in a way that also allows to trade-off other nanoscale objectives and constraints. The effective handling of vias, multiple metal layers and other aspects of nanoscale routing pose a series of algorithmic, implementation, benchmarking and integration challenges.

PART II

VLSI Placement

CHAPTER IV

Fine Control of Local Whitespace

At the 65nm technology node and below, many systematic manufacturing problems arise that can only be effectively mitigated in the physical design portion of the computer aided design (CAD) flow [13,66]. Issues such as parasitics variability induced by chemical mechanical polishing (CMP), yield loss due to increased shorts, via failures, forbidden pitches and forbidden polygonal shapes greatly affect yield. Techniques to handle these problems are known collectively as design for manufacturing (DFM), and are important to routing tools targeting 65nm designs [41].

An important factor in many DFM issues is design density, determined by local whitespace (also known as free space). Wire density is critical as too much wiring congestion has notable performance impact due to (i) longer wires resulting from routing detours (ii)

increased crosstalk which reduces reliability and degrades timing and (iii) increased via counts which lengthen signal propagation time and can decrease yield [134]. Conversely, too little wiring density will increase the likelihood that CMP will erode parts of wires, greatly increasing their resistance or leaving their connection open entirely [41]. While metal fill can mitigate harmful CMP effects, it does so at the cost of negative performance impact of additional crosstalk. Achieving the right balance of wire density through whitespace management avoids the performance impact of too much congestion while reducing the need for metal fill.

In addition to wire-density concerns, poor whitespace allocation during physical synthesis can increase total cell area due to buffer insertion and gate sizing [12]. One must reserve space locally to accommodate these operations to meet timing constraints, but reserving too much is wasteful [10]. In particular, a design that is placed too densely may have increased wirelength due to routing detours, and may be unable to close timing by inserting buffers into full regions of the chip. However, a design that is placed too sparsely will also have increased wirelength, and suffers both timing degradation and increased power consumption from more buffers. Therefore, accurate modeling of whitespace and precise cell density control are important concerns during the global and detailed placement phases of a physical synthesis flow.

In this chapter we propose several methods for top-down whitespace allocation to satisfy whitespace constraints. Our key contributions are:

- We introduce three user-controlled whitespace allocation techniques which are used to satisfy arbitrary density constraints in top-down global placement.

- We outline several detailed placement techniques which enforce density constraints while simultaneously improving interconnect length.

- We quantify the difficulty in satisfying density constraints and show why so many of the best solutions to the ISPD 2006 contest benchmarks [102] did not satisfy these constraints.

This chapter is organized as follows. Section 4.1 reviews traditional methods of whitespace allocation. Section 4.2 describes top-down whitespace allocation techniques and illustrates using them to satisfy density constraints in global placement. Whitespace allocation in detailed placement is discussed in Section 4.3. Section 4.4 provides empirical evaluation of our whitespace allocation techniques and we conclude in Section 4.5.

4.1 Traditional Whitespace Allocation

The literature includes several techniques to optimize whitespace distributions [5, 10, 24, 132]. A natural scheme for managing whitespace in top-down placement, uniform whitespace allocation, was introduced and analyzed in [24]. The authors derived expressions for the tolerance to be given to a min-cut partitioner such that whitespace would be allocated as uniformly as possible given the discrete nature of the problem.

A technique for non-uniform whitespace allocation presented in [5] adds disconnected standard cells to the design before placement using uniform whitespace allocation, and removes them immediately after. Care must be taken not to add too many cells to the design which complicates the work of many placement algorithms, increasing interconnect length or leading to overlapping circuit modules [49]. In [10], analytical methods

are used to allocate whitespace in sparse designs for min-cut placement. Before calls to partitioning, the design is placed quickly with an analytical algorithm. Cell area that is placed on either side of a proposed cut-line is used as an area target for min-cut partitioning. After floorplanning, [132] provides min-cost network-flow formulations to optimally redistribute whitespace in floorplans to reduce interconnect length.

There are relatively few techniques in the literature for respecting whitespace constraints imposed by a designer while still optimizing interconnect. Such constraints are helpful as they are typically imposed to improve routability, allow for effective buffer insertion, etc. In many cases, these constraints come in the form of cell density restrictions. One trivial way to ensure sparser cell densities in a placement is by artificially increasing cell sizes before placement (*bloating*) and shrinking them back to normal size afterward [124]. For the bloating to be effective, the majority of the original whitespace of the design must be consumed. This reduces the amount of whitespace available to the placer which is undesirable for reasons stated above. Bloating also makes density control in detailed placement more difficult as standard cells can only be bloated in discrete steps. Widening a standard cell by a single cell site often increases cell width by 20% or more whereas density control requires much finer precision, as seen in Table 4.1.

4.2 Top-down Whitespace Allocation

Top-down min-cut placement proceeds by successively dividing *placement bins*, the first of which contains the entire core area and all movable objects, until the bins are small enough to be optimally placed. Whitespace allocation is done per placement bin and in this section we describe three techniques: *uniform*, *minimum local* and *safe* whitespace

ALGORITHM 4.1: Per-bin whitespace allocation to satisfy density constraints

```
    ▷ Input: placement bin B, whitespace target targetWS
    ▷ Output: partitioned child bins
 1  SETTENTATIVECUTLINE(B)
 2  if (BINWHITESPACE(B) > targetWS and targetWS ≥ SAFEWS)
 3     then SETPARTTOLERANCESAFE(B, targetWS)
 4          CALLPARTITIONER(B)
 5          SHIFTCUTLINETOMAINTAINTARGETWS(B, targetWS)
 6     else if (BINWHITESPACE(B) > targetWS and targetWS ≥ MINLOCALWS)
 7             then SETPARTTOLERANCEMINLOCAL(B, targetWS)
 8                  CALLPARTITIONER(B)
 9                  SHIFTCUTLINETOMAINTAINTARGETWS(B, targetWS)
10             else SETPARTTOLERANCEUNIFORM(B)
11                  CALLPARTITIONER(B)
12                  SHIFTCUTLINETOEQUALIZEWS(B)
13  FINALIZECUTLINE(B)
14  CREATECHILDBINS(B)
```

Figure 4.1: Allocating whitespace in top-down placement to satisfy density constraints using uniform, minimum local and safe whitespace allocation.

allocation. Any one of these options can be chosen per bin based on the bin's whitespace and user-configurable options. Pseudocode in Figure 4.1 shows how these three techniques are used together to satisfy whitespace constraints.

Uniform Whitespace. If a bin has a user-defined "small" amount of whitespace or less, partitioning attempts to divide the cell area approximately in half, within a given tolerance. The appropriate partitioning tolerance is chosen based on whitespace deterioration and is calculated as described in Section 2.1.4. After a partitionment[1] is computed, the geometric cut-line for the bin is positioned so that each side of the cut-line has an equal percentage of whitespace. As tolerance is calculated assuming a fixed cut-line, the cut-line is shifted to make whitespace more uniform. Such whitespace allocation generally

[1]In this work, we use the word *partitionment* to refer to the solution of a partitioning instance, and *partitioning* as the act of finding such a solution. Other publications use the word *partitioning* to also refer to the solution; we use *partitionment* to disambiguate the concepts.

Table 4.1: Reallocation of whitespace in mPL6 [27] placements of selected ISPD 2006 contest benchmarks [102]. Local whitespace targets are the same as from the ISPD 2006 placement contest. Density violations are measured as the percentage of total cell area that violates density constraints. Using Capo 10.5 in ECO-system mode [117] in combination with our whitespace allocation techniques, we are able to significantly reduce the density violations of mPL6 placements.

Benchmark	Local WS Target	HPWL (e6)		Density Violation	
		Before	After	Before	After
newblue1	20%	66.61	74.37	1.039%	0.065%
newblue2	10%	199.05	223.87	3.701%	0.964%
newblue3	20%	283.40	297.97	2.813%	0.126%
newblue4	50%	293.22	304.99	7.238%	4.105%

produces routable placements, at the cost of increased wirelength.

Minimum Local Whitespace. If a bin has more than a user-defined minimum local whitespace (`minLocalWS`), partitioning will define a tentative cut-line that divides the bin's placement area in half. Partitioning targets an equal division of cell area, but is given more freedom to deviate from its target. Tolerance is computed so that with whitespace deterioration, each descendant bin of the current bin will have at least `minLocalWS`.

The assumption that the whitespace deterioration α in end-case bins is 0 made in [24] no longer applies, so the calculation of α must change. Since we want all child bins of the current bin to have `minLocalWS` relative whitespace, in particular end case bins must have at least `minLocalWS` and thus we may set the relative whitespace of an end-case bin $\overline{w}$ to be `minLocalWS` instead of a function of the partitioning tolerance τ. Using the assumption that α remains constant during partitioning, α is calculated as $\alpha = \sqrt[n]{\frac{\overline{w}}{w}}$ [24]. With the more realistic assumption that τ remains constant, τ is calculated as $\tau = \sqrt[n]{\frac{1-\overline{w}}{1-w}} - 1$ [24]. Knowing τ, $\alpha = (\tau + 1) - \frac{\tau}{w}$.

After a partitionment is calculated, the cut-line is shifted to ensure that `minLocalWS`

is preserved on both sides of the cut-line. If the minimum local whitespace chosen is small, the placer can produce tightly packed placements which improves wirelength.

Safe Whitespace. The last whitespace allocation mode is designed for bins with "large" quantities of whitespace. In safe whitespace allocation, as with minimum local whitespace allocation, a tentative geometric cut-line of the bin is chosen, and the target of partitioning is an equal bisection of the cell area. The difference in safe whitespace allocation mode is that the partitioning tolerance is much higher. Essentially, any partitioning solution that leaves at least `safeWS` on either side of the cut-line is considered legal. This allows for very tight packing and reduces wirelength, but is not recommended for congestion-driven placement.

4.3 Whitespace in Detailed Placement

Placement tools use several techniques to further reduce HPWL after global placement such as the sliding window optimizer RowIroning [23], but these techniques usually do not respect density constraints. To have finer control of whitespace than the sliding-window scheme, we present two detailed placement techniques that focus on whitespace allocation in addition to improvement of HPWL: a greedy cell-movement scheme and optimal whitespace allocation that preserves relative cell ordering by solving min-cost network-flow problems [17, 132].

Greedy Cell Movement. A gridded greedy movement technique can improve both wirelength and whitespace distribution. Pseudocode for our technique is shown in Figure 4.3. An arbitrary grid is imposed on the placement region to analyze local placement density. Density targets are set for each of the grid regions individually and can be non-

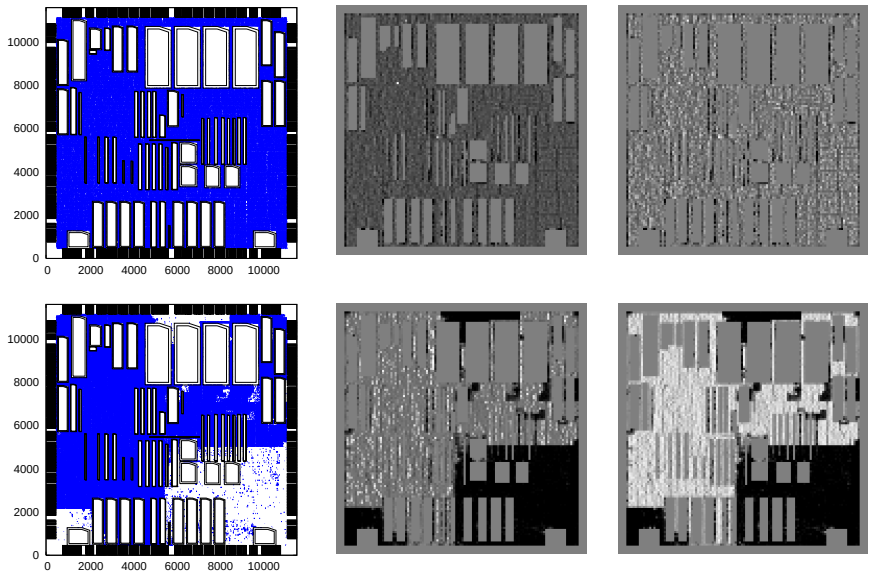


Figure 4.2: Column (a) shows Capo 10.5 global placements of the ISPD 2005 placement contest benchmark adaptec1 [104] with uniform whitespace allocation (top) and non-uniform whitespace allocation (bottom). Fixed obstacles are drawn with double lines. To indicate orientation, north-west corners of blocks are truncated. Columns (b) and (c) depict the local utilization of the uniform and non-uniform placements. Lighter areas of the placement signify regions that violate the target placement density whereas darker areas have utilization below the target. Areas with no placeable area (such as those with fixed obstacles) are shaded as if they exactly meet the target density. The target placement density for column (b) is 90% and the target for column (c) is 60%. Adaptec1 has 57.34% utilization. The HPWL for the uniform and non-uniform placements is 10.69e7 and 9.03e7 respectively. As the intensity maps show, when 60% utilization is the target, uniform whitespace allocation is much more appropriate than 12% minimum local whitespace. On the other hand, 12% minimum local whitespace is appropriate in terms of violations when the target is 90% utilization and has much better wirelength.

uniform. For standard cells that are in regions with density violations, location candidates are found in areas of lower density violation. Candidate moves are ranked by how well they alleviate violations as well as how they affect wirelength. We allow moves that increase HPWL, but only a fixed amount per move. Moves are made until a threshold of density improvement or a limit on increased HPWL is reached.

ALGORITHM 4.2: Greedy cell movement

```
    ▷ Input: initial placement P, grid of density targets G
    ▷ Output: optimized placement P
 1  bestImprovement ← 0, cellToMove ← −1
 2  foreach cell C
 3       do Search G for a legal location of C that best
            improves density violations and does not increase HPWL
            by more than HPWLLIMIT% (break ties using HPWL)
 4          Store the best location of C
 5          if (density improvement of C > bestImprovement)
 6            then bestImprovement ← density improvement of C
 7                 cellToMove ← C
 8  while (cellToMove ≠ -1)
 9       do Move cellToMove to its best location
10          Fix cellToMove
11          bestImprovement ← 0, newCellToMove ← −1
12          foreach cell C ≠ cellToMove
13               do if (best location of C overlaps with cellToMove)
14                    then Search G for the best location of C
15            else  if (C is connected to cellToMove)
16                    then Recalculate the density improvement of moving C to its best location
17                         if (density improvement of C ≤ 0)
18                           then Search G for the best location of C
19                    if (density improvement of C > bestImprovement)
20                      then bestImprovement ← density improvement of C
21                           newCellToMove ← C
22          cellToMove ← newCellToMove
23          if (density improvement of cellToMove < IMPROVEMENTCUTOFF)
24            then cellToMove ← −1
25          if (moving cellToMove to its best location increases HPWL beyond HPWLCAP)
26            then cellToMove ← −1
```

Figure 4.3: Greedy cell movement algorithm to reduce density violations while also taking HPWL into account.

A similar greedy movement technique can reclaim HPWL while leaving whitespace distribution unchanged. In this technique, pairs and triples of cells of approximately the same size are examined. The number of pairs and triples of cells in any modern design is intractable, so to keep runtime feasible our technique only considers pairs and triples of cells that are directly connected to each other by two-pin nets. After pairs and triples are collected, the HPWL gain is evaluated for swapping pairs of cells and the five non-trivial permutations of triples of cells. As the cells are of approximately the same size,

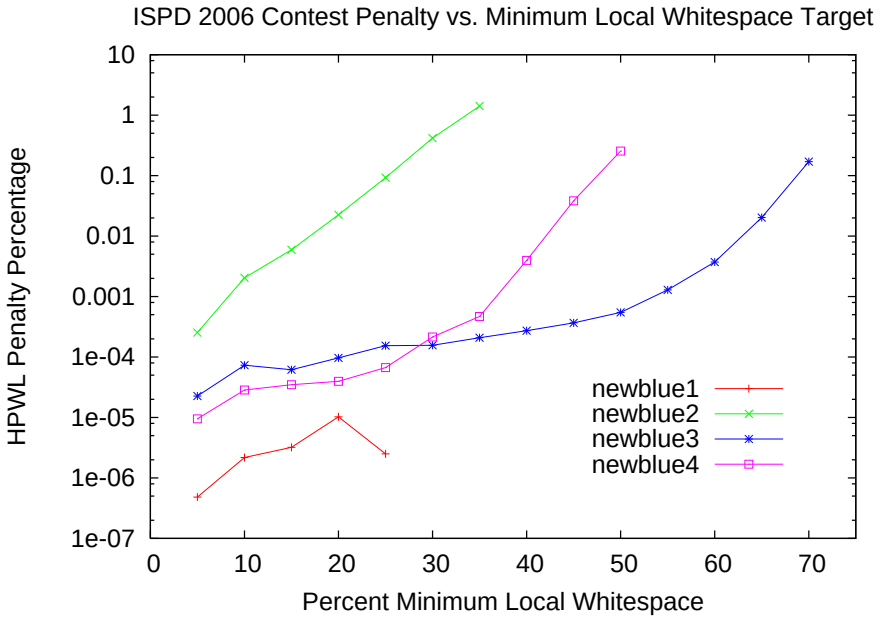


Figure 4.4: ISPD 2006 placement contest penalty for requested amounts of minimum local whitespace. The penalty is calculated based on the total amount of density-constraint violations. We test on benchmarks from the ISPD 2006 placement contest suite [102]. These benchmarks have 29%, 38%, 74% and 54% whitespace, respectively. Usually the penalty is very small when using our techniques (always less than 1.5%), but the penalty grows significantly as the requested whitespace approaches the amount of whitespace available in the design.

no overlap is produced by these swapping moves and whitespace distribution is largely unaffected. These cell-swapping moves are applied until a HPWL improvement threshold is reached.

Optimal Whitespace Redistribution. Optimal whitespace allocation in row-based placement [17] and floorplanning [132] given a fixed cell-ordering has previously been described in the literature. Unfortunately [17] only considered optimal whitespace allocation for the x-direction of a single row of a placement at a time while [132] was limited to relatively small floorplanning solutions generated using sequence-pairs. We extract the best of these techniques, and generate min-cost network-flow problems for generic floorplacement instances whose solutions optimally redistribute whitespace and snap cells to

Let the set of cells and macros be denoted:

$$C = c_1, c_2, \ldots, c_i$$

and the set of nets be denoted:

$$N = n_1, n_2, \ldots, n_j$$

Using the variables:

$$x_k \text{ for } 1 \leq k \leq i$$

$$L_k^x, U_k^x \text{ for } 1 \leq k \leq j$$

and constants:

$$left\_boundary, right\_boundary$$

$$placement(k).x \text{ for } 1 \leq k \leq i$$

$$width(k) \text{ for } 1 \leq k \leq i$$

we minimize:

$$\sum_{k=1}^{j} (U_k^x - L_k^x)$$

subject to constraints:

$$L_l^x \leq x_k \leq U_l^x \text{ where cell } k \text{ is on net } l$$

$$left\_boundary \leq x_k \text{ for } 1 \leq k \leq i$$

$$x_k + width(k) \leq right\_boundary \text{ for } 1 \leq k \leq i$$

$$x_k = placement(k).x \text{ where cell } k \text{ is fixed}$$

$$x_k + width(k) \leq x_l \text{ where cell } k \text{ is directly to the left of cell } l$$

Figure 4.5: Linear programming formulation (horizontal direction) to optimize HPWL of an existing placement. Further simplification is possible for two- and three-pin nets.

row/site boundaries.

Our technique builds upon the well-known linear programming formulations used, e.g., in [132] and [112] in that we impose linear constraints for movable objects based on their relative positions with respect to core boundaries and other movable objects. We include additional linear inequalities to account for fixed obstacles and region constraints. One major difference from previous work is that we guarantee that the x and y locations found align to legal sites and rows, as explained below.

We handle reallocation of whitespace separately for the horizontal and vertical directions, and preserve local relative ordering of movables in each direction. In other words, movable objects may not jump over each other or any fixed obstacles when whitespace is being reallocated. Unlike in global placement [112], we start with legal or nearly-legal locations. This simplifies our selection of relative constraints to include into the LP formulation as follows. In the horizontal case, we examine each row individually. For each cell or macro that intersects the row, we determine its immediate neighbors to the left and to the right (those objects with which the current object could feasibly overlap if it would slide to the left or right). These neighbors include movable objects, row or region boundaries as well as fixed obstacles. After the neighborhood relations are determined, we constrain an object to lie between its left- and right-hand neighbors. Construction of constraints for the vertical case is analogous where rows are replaced with columns and site width is replaced by row height. Lastly, to preserve global whitespace allocation characteristics, we add constraints to limit the amount of movement of any individual cell from its initial position. Unlike the formulation from [112], ours guarantees an overlap-free placement and needs to be solved only once. In contrast with [132], we include only several constraints per movable object rather than a quadratic number of constraints read from a sequence-pair. This significantly improves scalability and allows one to pack more tightly.

In addition to the constraints above, we minimize HPWL. This is done by adding L^x, U^x, L^y, U^y variables for each net, and the terms $(U^x - L^x)$ and $(U^y - L^y)$ to the objective function. The LP formulation for the horizontal case is enumerated in Figure 4.5. To solve the entire LP efficiently, we dualize it as in [132] and cast the dual as a min-

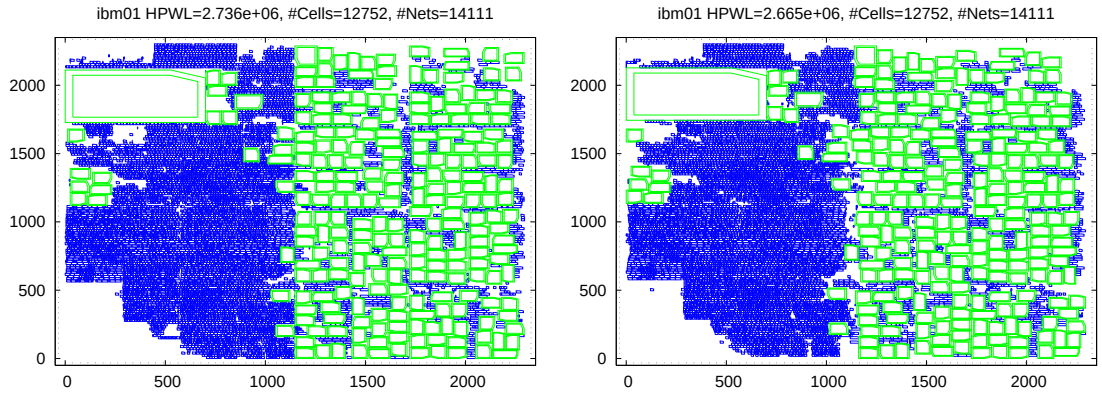


Figure 4.6: ICCAD 2004 IBM-MSwPins benchmark [1] ibm01 before (left) and after (right) optimal whitespace allocation via network flows. The HPWL improvement for this placement is 2.61% and takes only 10 CPU seconds to perform.

cost max-flow instance. The latter is solved using the scaling push-relabeling algorithm of Goldberg [63]. Nets from the original LP formulation become directed edges with unit capacity and zero cost in the dualized flow instance and distance constraints become directed edges with costs and unlimited capacity. Goldberg's implementation of push-relabeling in C uses integer variables for both costs and capacities. Thus the algorithm naturally produces integer solutions when the input is encoded in integers. We use this integrality to produce solutions that are row- and site-aligned — we scale coordinates so that integer x values correspond to legal sites and integer y values correspond to standard-cell rows. Thus our solutions need no further legalization.

Empirically this technique is extremely fast and provides non-trivial interconnect length improvement. Entire placements of up to 50000 cells can have their whitespace reallocated in 60 seconds or less. We have found that 50000 cells is a good trade-off between quality and runtime, so we break the placement area into a regular grid with a target of 50000 cells per grid cell and allocates whitespace in each region separately. We generally see interconnect length improvement of 2-3% with a runtime cost less than 10% of placement

Table 4.2: Relevant characteristics of select benchmarks from the IWLS 2005 suite [72]. "Grid size" is the size of the grid used for greedy cell movement.

Benchmark	# Movables	Grid size
aes_core	20795	14×14
ethernet	46771	30×30
mem_ctrl	11440	12×12
pci_bridge32	16816	17×17
usb_funct	12808	14×14
vga_lcd	124031	41×41

runtime (see Section 4.4 for more detailed results). Figure 4.6 depicts a placement of the mixed-size design ibm01 (from the ICCAD 2004 IBM-MSwPins benchmarks [1]) before (left) and after (right) whitespace optimization with flows.

4.4 Empirical Results

We have implemented proposed whitespace allocation techniques in the open-source placer Capo 10.5[2] [115]. In this section we evaluate our techniques in the contexts of satisfying density constraints and optimizing HPWL on a wide variety of publicly available benchmarks.

Whitespace Reallocation. We combine our whitespace allocation techniques with the ECO-system [117] mode of Capo 10.5 to reallocate whitespace in ISPD 2006 contest solutions from the mPL6 placer [27]. mPL6 uses a multilevel analytical technique for global placement with cell bloating to help meet target densities [47] and the XDP detailed placer [52] which legalizes and applies sliding window techniques to recover wirelength. At the ISPD 2006 placement contest, mPL6 produced the best solutions when not consid-

[2]Our use of Capo as an implementation platform is justified by Capo's competitive results on difficult mixed-size instances [107] and all routability-driven placement benchmarks reported in the literature. Capo's routed solutions have best published via counts [116], which is very important for DFM and yield. Vias also significantly impact timing, and may complicate routing by blocking routing tracks [134].

Table 4.3: Correction of local density violations by greedy cell movement techniques. Benchmarks are selected from the IWLS 2005 benchmark suite and each have 38% total whitespace [72]. Density violations are measured as the percentage of total cell area that violates density constraints. Greedy cell movement corrects all density violations when requested local whitespace is 25% or less and in many cases improves HPWL as well.

Benchmark	Local WS Target	Density Violation		HPWL Impact	Runtime (s)
		Before	After		
aes_core	25%	0.042%	**0%**	**-1.699%**	43
aes_core	30%	0.208%	0.017%	+0.513%	4
aes_core	35%	0.572%	0.246%	+0.530%	8
ethernet	20%	0.002%	**0%**	**-0.534%**	163
ethernet	25%	0.036%	**0%**	+0.501%	174
ethernet	30%	0.113%	0.015%	+0.503%	111
ethernet	35%	0.892%	0.459%	+0.522%	128
mem_ctrl	30%	0.008%	**0%**	**-0.023%**	3
mem_ctrl	35%	0.586%	0.110%	+0.518%	4
pci_bridge32	20%	0.002%	**0%**	**-1.167%**	25
pci_bridge32	25%	0.061%	**0%**	**-1.182%**	31
pci_bridge32	30%	0.010%	0.003%	+0.530%	11
pci_bridge32	35%	0.823%	0.331%	+0.506%	17
usb_funct	30%	0.030%	**0%**	**-0.547%**	16
usb_funct	35%	0.356%	0.068%	+0.578%	6
vga_lcd	20%	0.0002%	**0%**	**-0.440%**	622
vga_lcd	25%	0.045%	**0%**	+0.132%	743
vga_lcd	30%	0.264%	0.101%	+0.500%	585
vga_lcd	35%	1.288%	0.758%	+0.500%	740

ering runtime, but as shown in Table 4.1 the solutions did not satisfy the density constraints imposed by the competition. These density violations can be significantly improved using our technique, but only at the cost of significantly increased wirelength.[3] In the smaller benchmarks, `newblue1` and `newblue2`, the cost in HPWL is approximately 12%. On `newblue3` and `newblue4` the increase in HPWL is much lower at 5% and 4%, respectively. This shows, especially on the larger benchmarks, that density violations can

[3]It is important to note that the coefficients in the ISPD 2006 penalty formula were chosen rather arbitrarily, while the effective cost of violations greatly depends on the types of problems caused by violations, such as increased crosstalk noise and need for DFM fix-ups.

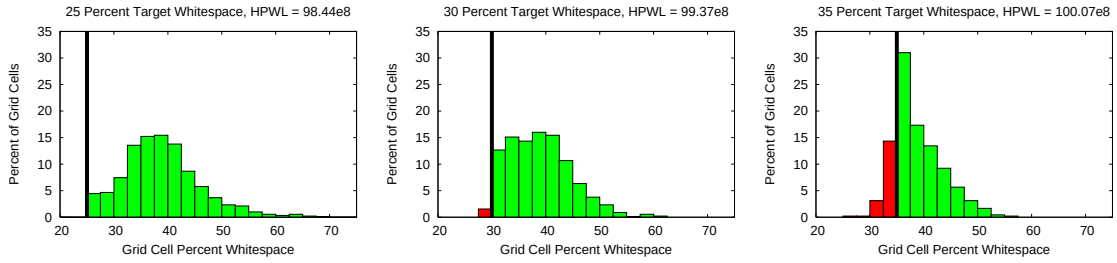


Figure 4.7: Controlling whitespace distribution on the `ethernet` benchmark from the IWLS 2005 benchmark suite [72], which has approximately 38% whitespace. We divide the placement area into a regular grid and report whitespace distribution across grid cells when targeting (a) 25%, (b) 30% and (c) 35% minimum local whitespace. As the minimum whitespace requested approaches the total whitespace, the constraint becomes more difficult to satisfy, but our techniques are successful in producing solutions that are legal or nearly-legal for the majority of grid cells.

be improved dramatically with a reasonable increase to HPWL. These reallocated placements outperform all but one placer on one benchmark from the ISPD 2006 contest (only Dragon's placement of `newblue4` has a lower density penalty) and have extremely competitive HPWL.

Density Constraint Satisfaction. We implemented all of our proposed whitespace allocation techniques in the Capo 10.5 framework and test uniform and non-uniform whitespace allocation on the ISPD 2005 contest benchmark adaptec1 (57.34% utilization) with 60% and 90% target whitespace densities. The HPWL for the uniform and non-uniform placements is 10.69e7 and 9.03e7, respectively. Uniform whitespace produces almost no violations when the target is 90% and relatively few when the target is 60%. The non-uniform placement has more violations when compared to the uniform placement, especially when the target is 60%, but remains largely legal with a 90% target density. Hence, uniform whitespace allocation is appropriate when target density is near the total amount

of whitespace in a design, otherwise non-uniform allocation can be used to improve wirelength.

In Figure 4.7, we show histograms of grid cell densities across the `ethernet` benchmark [72] when given local whitespace constraints of (a) 25%, (b) 30% and (c) 35%. The `ethernet` design has 38% total whitespace and our techniques are able to achieve completely legal solutions when 25% local whitespace is requested, but the constraints become more difficult to satisfy at 30% and 35%. Despite the difficulty, our techniques produce solutions that are legal or nearly-legal for the vast majority of grid cells.

The inherent difficulty in satisfying minimum whitespace constraints as the requested whitespace approaches the total amount of whitespace in the design is also apparent from Figure 4.4. Here we place selected benchmarks from the ISPD 2006 placement contest with a wide range of requested local whitespace values. We use our whitespace allocation methods to match the requested amount of local whitespace for each of the instances and evaluate the legality of our solutions with the ISPD 2006 placement contest density penalty function [102]. The penalty is calculated based on the total amount of density-constraint violations in the placement. Our solutions generally have very small penalties (always less than 1.5%) suggesting that our techniques satisfy density constraints well. Note how the density penalty grows more quickly as the amount of requested whitespace approaches the total amount of whitespace in the design.

Greedy Cell Movement. Table 4.3 shows the effectiveness of greedy movement techniques in removing density violations. Benchmarks are selected from the IWLS 2005 benchmark suite and each have 38% total whitespace [72]. Size characteristics of these

Table 4.4: HPWL improvement due to flow-based whitespace redistribution on the ICCAD 2004 IBM-MSwPins mixed-size benchmarks [1]. On average, the flows are able to reallocate whitespace and improve HPWL by nearly 3% while scaling well with increasing quantities of movable objects.

Benchmark	# Movables	Runtime (s)	Improvement
ibm01	12506	11	2.41%
ibm02	19342	23	2.60%
ibm03	22853	27	3.81%
ibm04	27220	38	2.96%
ibm05	28146	46	2.27%
ibm06	32332	44	2.87%
ibm07	45639	69	3.17%
ibm08	51023	64	2.82%
ibm09	53110	69	4.23%
ibm10	68685	129	1.91%
ibm11	70152	102	4.42%
ibm12	70439	135	1.87%
ibm13	83709	127	3.83%
ibm14	147088	262	2.64%
ibm15	161187	342	3.67%
ibm16	182980	404	2.62%
ibm17	184752	446	2.54%
ibm18	210341	338	2.47%
Average			2.86%

benchmarks are shown in Table 4.2. Density violations are reported as the percentage of total cell area that violates density constraints. Greedy cell movement corrects all density violations when requested local whitespace is 25% or less and in many cases improves HPWL as well. As the requested local whitespace approaches the total whitespace, greedy movement is not able to remove all of the density violations without making HPWL increase more than 0.5%. With a higher limit on HPWL increase, greedy movement can apply more moves and further reduce density violations.

Flow-based Whitespace Redistribution. We test optimal whitespace redistribution based on min-cost network flows on the ICCAD 2004 IBM-MSwPins benchmark suite [1]. Table 4.4 gives detailed runtime and HPWL improvement results for each of the

IBM-MSwPins benchmarks. We do not provide overflow statistics on these designs since our flow-based whitespace redistribution maintains global whitespace characteristics. On average, HPWL on these benchmarks is improved by 2.86% and runtimes scale nearly linearly with benchmark size. Figure 4.6 depicts a placement of the ibm01 mixed-size design before (left) and after (right) whitespace optimization with flows.

4.5 Conclusions

In this chapter we have introduced methods for satisfying whitespace constraints in top-down placement while also optimizing interconnect. These constraints take the form of cell density limits on a placement. A follow-up to the ROOSTER work on routability-driven placement [116] has found that cell density limits can be extremely useful for promoting routability, decreasing metal fill, improving yield, etc. [35]. Our techniques consistently improve the quality of whitespace allocation of top-down as well as analytical placement methods and achieve low penalties on designs from the ISPD 2006 placement contest with minimal interconnect increase.

CHAPTER V

Routability Optimization in Placement

We observe that one major source of sub-optimality in modern standard-cell placement is the optimization of half-perimeter wirelength (HPWL) rather than objectives which more closely model concerns seen in routing. Our main contribution is a series of optimization techniques for Steiner-tree wirelength (StWL) in global and detailed placement without a significant runtime penalty, making the use of half-perimeter wirelength (HPWL) unnecessary. We draw on recent results in min-cut placement, particularly improvements [37] of the terminal propagation technique [123], which better correlate minimizing net-cut with minimizing HPWL. We generalize this technique and show that with adequate data structures it reduces StWL in global placement efficiently. We leverage recent research on fast and accurate construction of Steiner trees to make StWL minimization in placement practical.

To our knowledge, minimization of StWL in min-cut bisection has not been attempted before, particularly the net-vector technique [68] cannot capture Steiner-tree lengths in bisection or quadrisection (for more details see Section 5.1.1). There has also been work in weighting the HPWL of individual nets based on their pin counts [39]. Later work im-

Table 5.1: Objectives of the Place-and-Route process and how they compare with objectives of placement techniques. Traditional work on placement does not optimize or even report the objectives most pertinent for Place-and-Route. It is particularly difficult to optimize objectives that are measured *relative* to a given industrial router. We improve key objectives by departing from traditional HPWL optimization. Optimizing congestion estimates *per se* appears of limited use.

Objectives/constraints in Place-and-Route		Use in placement			Our empirical improvements
		Pertinent	Popular	Ours	
Relative	Routability	*			+
	Routed WL	*			+
	Via count	*		limited	+
	Timing	*	$\sim$	potential	
	Dynamic power	*		potential	
	Router runtime	*			+
Absolute	Congestion estimates	?	*	*	+
	Placer runtime	*	*	limited	-
	Steiner-tree WL			*	+
	HPWL		*		-

proved on these weighting techniques [20]. The authors of [21] find that these weighted wirelength techniques are reasonable predictors of routed wirelength, but that smaller weighted wirelength can translate into larger routed wirelength making the use of weighted wirelength as an optimization "questionable."

Our Steiner-tree driven detailed placer leverages the speed of the recent FLUTE package [44]. The closest work in detailed placement [73] models single-trunk Steiner trees to reduce congestion in FPGAs. While effective, this technique requires exorbitant amounts of runtime. Instead, our detailed placer considers optimal Steiner trees and is quite fast.

We also build upon recent work in congestion-driven placement that uses congestion maps. In [143], congestion maps are built after global placement, and annealing moves are applied to minimize a congestion metric. Another technique, known as WSA [93], is applied after detailed placement. It identifies areas with high congestion and injects whitespace into these areas in a top-down fashion. Our work uses congestion maps from

[141] to allocate whitespace in a manner similar to WSA but *proactively, during global placement*. As a result, our placer ROOSTER (Rigorous Optimization Of Steiner-Trees Eases Routing) produces the best known routed wirelengths on the IBMv2 benchmarks [143].

At the 90nm technology node and below, increased via resistance, manufacturing variability and manufacturing defects require unprecedented attention to vias. In particular, via resistance may vary by more than other important circuit parameters — in some technologies a difference of 30 times has been observed between neighboring vias. Therefore, manufacturers prefer and sometimes require vias to be doubled, since this averages out the variation. To this end, we point out that a range of easy-to-implement detailed placement algorithms (those of the cell-shifting variety) tend to increase via counts, even when they improve routability. ROOSTER avoids them and exhibits the smallest via counts on standard benchmarks among all published results and our runs of recent placement tools.

In the remainder of this chapter, Section 5.1 describes previous work on routing-driven VLSI placement. Section 5.2 discusses choosing the right objective to optimize in placement and outlines a first implementation in floorplanning. Sections 5.3 and 5.4 introduce the realization of Steiner-tree modeling in min-cut placers and Steiner-driven detailed placement, respectively. Section 5.5 outlines whitespace allocation to improve routability. Experimental results are given in Section 5.7, and Section 5.8 concludes and motivates further applications of our techniques.

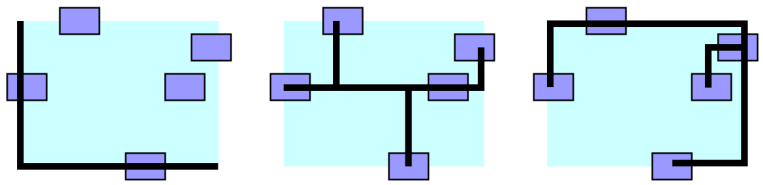


Figure 5.1: HPWL (left), Steiner WL (center) and Rectilinear Minimal Spanning Tree (MST) WL (right) for a five-pin net.

5.1 Previous Work on Routability-driven Placement

Traditionally, placement and routing are treated as two independent optimization problems. Standard-cell placement is generally seen as the problem of finding non-overlapping row- and site-aligned positions for cells while minimizing the wirelength of the design. Currently, HPWL is the estimate of choice for wirelength minimization in placement because it is computationally easy and exactly calculates Rectilinear Steiner Minimal Tree (RSMT) length for two- and three-pin nets. Unfortunately, routers construct routed wires using Steiner trees whose length is under-approximated by HPWL. Figure 5.1 shows how HPWL, RSMT, and Minimal Spanning Tree (MST) length differ for a given five-pin net. Note that the shortest vertical segment in the RSMT is not included in the HPWL of the net. Since RSMT construction is an NP-complete problem [60], it has been generally regarded as too computationally demanding for use in placement [68]. To illustrate how a placer optimizes its chosen objective, we describe a specific technique – top-down min-cut placement.

5.1.1 Routability-driven Top-down Min-cut Placement

Top-down placement algorithms seek to decompose a given placement instance into smaller instances by subdividing the placement region, assigning modules to subregions

and cutting the netlist hypergraph [21]. Min-cut placers generally use either bisection or quadrisection to divide the placement area and netlist. Netlist division is commonly implemented with the Fiduccia-Mattheyses heuristic and derivatives [22,59], or alternately with quadratic placement and geometric partitioning [16].

Placement bins. Each hypergraph partitioning instance is induced from a rectangular region, or bin, in the layout. In this context a *placement bin* represents (i) a placement region with allowed module locations (*sites*), (ii) a collection of circuit modules to be placed in this region, (iii) all signal nets incident to the modules in the region, and (iv) fixed cells and pins outside the region that are adjacent to modules in the region (*terminals*). Top-down placement can be viewed as a sequence of passes where each pass examines all bins and divides some of them into smaller bins. These smaller bins collectively contain the entire layout area and cells of the original instance. When placement bins are divided, careful choice of vertical or horizontal cut direction influences wirelength and routing congestion in resulting solutions [131].

Using multi-way partitioning. In an attempt to improve basic recursive bisection, many researchers have noted that it eventually produces multi-way partitions which could be alternatively achieved by direct methods using wirelength-like multi-way objectives. In [68], the authors make use of quadrisection and show how several different cost functions other than cut can be optimized efficiently, although with overhead greater than that of bisection. One such cost function is the Minimum Spanning Tree (MST) length which they note is a far more accurate predictor of routed wirelength than net-cut. The authors note that in order for a wirelength evaluator to be feasible for placement optimization,

it must have evaluation complexity equal to or less than MST. On the other hand, the authors claim that their techniques can apply to "arbitrarily complicated per-net placement objectives" [68].

The net-vector technique includes the computation of 2^p integer costs per optimization objective defined for p partitions ($p = 4$ in [68] because quadrisection is used). It then looks up these costs during partitioning. Unfortunately, such look-ups require the discretization of pin locations and cannot account for the location of fixed terminals with as much precision as our work. Furthermore, the Steiner-tree objective on a discretized 2x2-grid does not differ from the discretized MST objective, hence it appears that optimizing StWL would require at least 16-way partitioning with large net-vector tables. However, no 16-way *geometric* partitioners can be found in the literature that are competitive with recursive bisection. In our work, Steiner trees are built on the fly for each configuration, but the overall runtime remains reasonable.

Cell bloating. BonnPlace [16] presented temporary standard cell bloating for use during partitioning-based placement. Each cell is expanded based on the magnitude of the congestion estimated for the region where the cell is currently located, the number of pins on the cell and a user-specified constant. After each call to partitioning, BonnPlace estimates congestion, inflates cells in all over-congested regions, and then re-partitions the design [16]. BonnPlace techniques can only be applied during top-down placement rather than after; it is unclear how to apply them during global placement other than by partitioning.

5.1.2 Estimating Congestion and Routed Wirelength

Congestion Maps. There have been many recent advances in estimating routing congestion. Most have come in the form of more accurate and faster *congestion maps* [83, 141]. A congestion map is a two-dimensional representation of one or more layers of the routing grid. In previous work, the routing edges from all layers are collapsed into a single layer to facilitate estimation of congested areas. For each two-pin net in a probabilistic congestion map, all shortest length routings of the net are assigned equal weight and fractions of a net are added to routing edges through which they could pass. Nets with 3 or more pins are typically divided into pairs of pins by constructing minimum spanning trees or Steiner trees [141].

In this work, we make use of the congestion mapping techniques presented in [141] which assumes that routers attempt to route nets with the fewest bends possible. The technique models two-pin nets in only L and Z shapes, unlike other methods that consider all possible shortest paths between two pins equally. Empirically, the authors of [141] have found that some routers are able to find routes with one bend 60% of the time and two bend routes for the majority of other nets. Thus, one-bend and two-bend routes are weighted this way in their maps. Empirical results show that such estimates correlate well with actual routing usage in the Magma Place-and-Route flow [141].

Rectilinear Steiner Minimal tree evaluators. The problem of constructing Rectilinear Steiner Minimal trees is known to be NP-hard [60]. Specifically, it is the problem of connecting a given set of points in the Manhattan plane by a minimum-length tree, which can use additional branching (Steiner) points. This problem admits polynomial-time ap-

proximations and practical heuristics. Three such algorithms with available source code are Batched Iterated 1-Steiner (BI1ST) [81], FastSteiner [78], and FLUTE [44]. BI1ST, albeit the oldest and slowest of these algorithms, generally produces the best solutions overall. FLUTE, the most recent and fastest algorithm, is provably optimal for instances with 9 or fewer points. FastSteiner falls in the middle in terms of speed and solution quality.

5.1.3 Achieving Routable Placements

It is well-known that a placement with small HPWL may be unroutable due to uneven routing demand and ensuing wiring congestion. For this reason, modern placers must explicitly account for routing congestion in order to produce routable placements. In [143], congestion maps are built after global placement, and annealing moves are applied to minimize a congestion metric. Another technique known as WSA [93] is applied after detailed placement. WSA uses congestion maps to identify areas with high congestion and injects whitespace into these areas in a top-down fashion. After whitespace allocation, cells typically overlap each other and legalization is required. After legalization, window based detailed placement techniques are applied to reduce wirelength that was increased during whitespace allocation and legalization. Cell bloating [16, 124] and cell spreading [93] are used to tie whitespace to specific cells, rather than regions as in techniques based on congestion maps.[1]

Congestion-driven analytical placement. It is well-known that improving the half-perimeter wirelength (HPWL) targeted by most placers sometimes complicates the work

[1]Cell bloating artificially increases the *width* of cells because their heights are determined by rows. However, the peak demand for horizontal tracks does not decrease because cells are not spread vertically. To the contrary, by spreading cells horizontally cell bloating increases the overall demand for horizontal tracks.

of routers and results in unroutable placements [16, 93, 144]. To this end, the placers that performed best at the ISPD 2006 contest — NTUplace and KraftWerk2 — have recently been enhanced to mitigate congestion [75,126] blending quick congestion estimates into the objective function. The unnamed technique presented in [135] estimates wiring density without using a router (and thus does not estimate the impact of detouring) and incorporates minimization of these estimates within an analytical placement engine. A related extension to FastPlace [45] goes further and embeds a fast global router into the placement loop. It demonstrates that the same router produces shorter routes starting from enhanced FastPlace placements.

These approaches are quite different, and the use of congestion estimates is much easier to implement. Also, the evidence in [16, 75, 126, 135] is more convincing because it involves complete commercial routers and confirms violation-free completion. However, our experience with large industry ICs suggests that congestion estimates around obstacles and blockages are often very inaccurate. This observation does not conflict with results in [16, 75, 126, 135] because the benchmarks used there do not contain significant blockages and, in any case, the routing information in those benchmarks was generated artificially. While routing congestion is known to impact circuit timing, these effects were not discussed in previous academic publications.

5.2 Choosing the Proper Objective

In this section we seek a wirelength estimator that adequately captures routed wirelength and is suitable for efficient optimization. While the former appears within reach, the latter turns out more difficult.

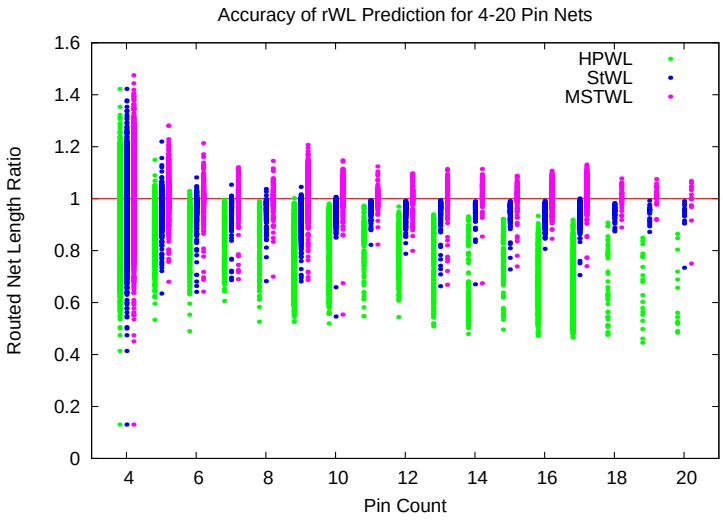


Figure 5.2: Comparing the accuracy of routed wirelength (rWL) estimators HPWL (left lines), StWL (middle) and MST WL (right) for nets with 4-20 pins in the `vga_lcd` design from the IWLS 2005 benchmarks [72]. StWL was calculated using FastSteiner [78].

5.2.1 Estimating Net Length

A priori wirelength estimation is the subject of extensive literature [20]. In this work we are mainly interested in evaluating and using simple per-net estimators, such as weighted HPWL, identified previously as a reasonable compromise between HPWL and Rectilinear Steiner Minimal Tree (RSMT) evaluators [20]. However, experiments described in [21] reveal poor correlation between total weighted HPWL and total routed WL in placement. Therefore, we do not consider weighted HPWL as a potential objective in our work.

On the positive side, recent progress on fast RSMT evaluators [44, 78] opens the possibility of using them in optimization. HPWL and RSMT WL (aka Steiner WL) share the same drawback — they both underestimate routed wirelength (rWL), due to detours, pin access problems, etc. A common response to this issue is to use the Minimal Spanning Tree length (MSTWL) [68]; this is relatively easy to compute and does not exceed Steiner

WL by more than 50%. Therefore, we also include MSTWL in our experiments.

To test our intuition, we perform the following experiment. We analyze a placement of the `vga_lcd` design from the IWLS 2005 series of benchmarks [72] which was routed without violation by Cadence WarpRoute. The `vga_lcd` design has 124,031 standard cells and 124,098 nets. For each net with 4-20 pins, we plot the ratios of HPWL, StWL and MSTWL (length of the MST of the net) to routed net length vs. the pin count of the net. See Figure 5.1 for a comparison of HPWL, StWL and MSTWL for a five-pin net. StWL was calculated using FastSteiner [78]. Statistics for two- and three-pin nets are not shown as HPWL and StWL produce identical numbers. For each net, three values are plotted in Figure 5.2: $\frac{HPWL}{rWL}$ (left), $\frac{StWL}{rWL}$ (middle) and $\frac{MSTWL}{rWL}$ (right). Nets are separated by their pin counts. In some cases, HPWL and StWL have ratios greater than 1.0. This is due to routers making use of internal wiring within cells that does not count toward reported wirelength. The discrepancy is exacerbated by wide pins present in many cell libraries, as well as by logically and electrically equivalent pins.

Figure 5.2 shows that HPWL is a poor estimator of routed net length — it can significantly under-estimate rWL and includes a great amount of noise since the range of ratios to rWL is large. As one might expect, StWL typically underestimates routed net length as well, but its range of ratios in the figure is significantly smaller than for MST. This means that with a proper correction, StWL may be a more accurate estimate than MST. [2] More importantly, given two nets, StWL estimates can predict more reliably which net will have longer routed length, i.e., StWL has higher *fidelity*. Further experiments described in Sec-

[2]Figure 5.2 suggests that MST is the most accurate estimator of routed net length *on average* for the router used on this design because the ranges of ratios for MST are centered at 1.0.

Table 5.2: Fixed-outline floorplanning to minimize HPWL versus Steiner WL. All StWLs were calculated using the Steiner evaluator FLUTE [44]. All wirelength and runtimes are averaged over 50 runs. Optimizing Steiner WL increases runtime by a minimum of 2.43x for `n300` and a maximum of 29.53x for `ami33`.

Bench-mark	#Macros	#Nets	Max Edge Degree	Avg Edge Degree	#Nets with Degree > 3
ami33	33	123	34	3.4797	8
ami49	49	408	24	2.2892	19
n10	10	118	4	2.1017	2
n30	30	349	3	2.0716	0
n50	50	485	4	2.1650	1
n100	100	885	4	2.1164	5
n300	300	1893	6	2.3022	47

Bench-mark	Minimizing HPWL			Minimizing Steiner WL		
	HPWL	StWL	Time (s)	HPWL	StWL	Time (s)
ami33	83267	105857	1.20	83434	103566	35.44
ami49	913680	934291	2.90	932408	951646	13.67
n10	56767	56841	0.12	57169	57277	0.45
n30	172614	172614	1.07	170527	170527	3.78
n50	204061	204100	3.16	207151	207193	9.70
n100	339423	339545	12.76	340396	340502	37.05
n300	764859	766389	122.98	760575	761968	299.32
Ratio	1.000	1.000	1.000	1.004	1.001	4.590

tion 5.6 have shown that the fidelity of net length estimates, rather than their accuracy is key in placement. Indeed we have independently verified using MSTWL as an optimization objective is worse than StWL for routability and may be less effective than HPWL in certain situations (see Table 5.13 and discussion in Section 5.7).

5.2.2 Impact of Steiner-tree Evaluation

As a first attempt at optimizing Steiner WL, we replaced the HPWL subroutine of the fixed-outline annealing-based floorplanner Parquet with FLUTE [44], a very fast Steiner-tree evaluator. The choice of floorplanning for this experiment is explained by its relative simplicity. It also clearly illustrates the impact of optimizing Steiner length on runtime

and solution quality in circuit layout.

Table 5.2 shows the netlist statistics for some common floorplanning benchmarks as well as runtimes and wirelengths with and without the use of FLUTE. All runtimes and wirelengths are averages over 50 runs. As is evident from the table, blindly replacing an HPWL evaluator with a Steiner-tree evaluator, even one as fast as FLUTE, can result in a huge increase in runtime when nets have nontrivial pin count. Trivial pin count for any Steiner evaluator is three or fewer since Steiner length is the same as HPWL in such instances. All the nets in the `n30` benchmark have trivial pin count, but we observe a 3.53x increase in runtime. The reason for this runtime increase is that calling a Steiner-tree evaluator requires nontrivial overhead (most notably the removal of duplicate points which requires sorting) as compared to Parquet's HPWL evaluator which is hand-tuned for speed [25].

The data in the table is also quite striking in that it shows that optimizing for Steiner length was not particularly effective, as Steiner wirelength and HPWL were both increased across all of the benchmarks. This shows that what one may think is an obvious method to reduce Steiner wirelength may not be all that useful. One possible explanation of this strange result is that Steiner WL is not a convex objective. Thus, it may require a longer annealing schedule than a convex objective like HPWL, whereas in our experiments the annealing schedule was fixed.

Our empirical results suggest that Simulated Annealing is not compatible with Steiner WL evaluation as Simulated Annealing relies on frequent net length computation, making Steiner WL calculation the bottleneck. Furthermore, Simulated Annealing appears to be

ineffective in optimizing Steiner WL as Steiner WL increased on average in our experiments. We pursue a different approach and, surprisingly, manage to optimize Steiner WL with only a modest runtime penalty.

5.3 Minimizing Steiner-tree Length in Global Placement

In this section, we describe new techniques to minimize Steiner wirelength in min-cut placement. In addition to the overall methods that make minimizing Steiner wirelength possible, we present data structures new to min-cut placement that keep runtimes practical. These global placement techniques alone can reduce routed wirelength by up to 7%, as demonstrated in Figure 5.7.

A framework for minimizing StWL. To minimize total StWL during min-cut placement, we capture it using the weighted net-cut objective used in partitioning. In the case of HPWL minimization, this has been accomplished in [123] with a 7-case analysis. A different group reduced this technique to the calculation of three wirelengths per net when building a partitioning instance and verified resulting empirical improvements [37]. To be clear, the three wirelengths that must be calculated per net (w_1, w_2 and w_{12}) completely determine the connectivity and costs of all nets in the derived partitioning hypergraph [37].

While the formulation from [37] is more compact than the one from [123], we also note that it is far more general. For each net in a partitioning instance, one must calculate the cost of all nodes on the net being placed at the center of partition 1 (w_1), the cost of all nodes on the net being placed at the center of partition 2 (w_2) and the cost of all nodes on the net being split between the centers of partitions 1 and 2 (w_{12}). For each net of the netlist hypergraph relevant to the partitioning instance, two nets are created in the

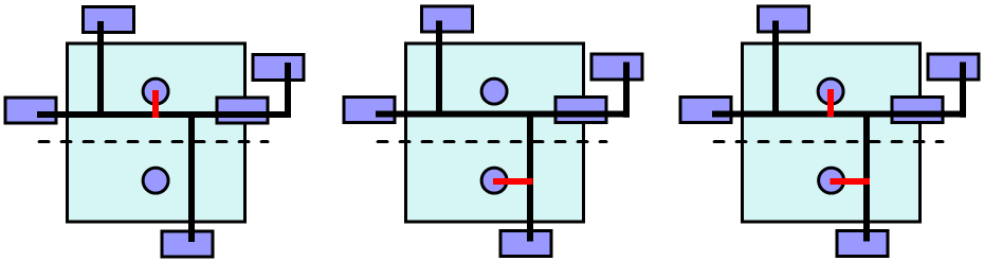


Figure 5.3: Calculating the three costs for weighted terminal propagation with StWL: w_1 (left), w_2 (middle), and w_{12} (right). The net has five fixed terminals: four above and one below the proposed cut-line. For the traditional HPWL objective, this net would be considered inessential. Note that the structure of the three Steiner trees may be entirely different, which is why w_1, w_2 and w_{12} must be evaluated independently.

partitioning hypergraph: one with weight $w_{12} - max(w_1, w_2)$ and the other with weight $|w_1 - w_2|$ [37]. The new net with weight $w_{12} - max(w_1, w_2)$ connects all of the movable objects (non-terminals) of the original net. The new net with weight $|w_1 - w_2|$ connects all of the movable objects of the original net to one of the fixed terminals in either partition 1 or 2. This new net connects to the terminal in partition 2 when $w_1 > w_2$ and to the terminal in partition 1 when $w_1 < w_2$. If either net has weight 0, it is discarded from the problem. The authors of [37] show, assuming $w_{12} \geq max(w_1, w_2)$, that this net weighting scheme makes minimizing HPWL equivalent to minimizing the weighted net-cut of the partitioning hypergraph.

The points required to calculate w_1 for a net are the positions of the terminals on the net plus the center of partition 1. Similarly, the points required to calculate w_2 are the positions of the terminals plus the center of partition 2. Lastly, the points to calculate w_{12} are the positions of the terminals on the net plus the centers of both partitions. See Figure 5.3 for an example of cost calculation. Clearly, the StWL of the set of points necessary to calculate w_{12} is at least as large as that of w_1 and w_2 since it contains an additional point.

Since StWL satisfies the assumptions made by the authors of [37], weighted partitioning can be used to minimize StWL. To our knowledge, such a framework has not been known in min-cut placement until now.

The simplicity of this framework for minimizing StWL is deceiving. In particular, the propagation of terminal locations to the current placement bin and the removal of inessential nets [23] — standard techniques for HPWL minimization — cannot be used when minimizing StWL. Moving terminal locations drastically impacts Steiner-tree topology and can make StWL estimates poor. Nets that are considered inessential in HPWL minimization are not necessarily inessential when considering StWL because there are many Steiner trees of different lengths that have the same bounding box. Figure 5.3 illustrates a net that is inessential for HPWL minimization but essential for StWL minimization.

Pointsets with multiplicities. Building Steiner trees for each net during partitioning is a computationally expensive task. Table 5.2 in Section 5.2.2 shows how expensive a naive replacement of HPWL with Steiner-tree evaluation can be in floorplanning. Even traversing nets to collect all relevant point locations when building Steiner trees can be very time-consuming. Therefore, the main challenge in supporting StWL minimization is to develop efficient data structures and limit additional runtime during placement.

To keep runtime reasonable when building Steiner trees for partitioning, we propose a simple yet highly effective data structure — *pointsets with multiplicities*. For each net in the hypergraph, we maintain two lists. The first list contains all the unique pin locations on the net that are fixed. A fixed pin can represent terminals, and fixed and placed objects in the core area. The second list contains all the unique pin locations on the net that are

movable, i.e., all other pins that are not on the fixed list. We maintain a unique list of points so that we don't pass any redundant points to Steiner evaluators which may increase their runtime. To do so efficiently, we keep the lists sorted. For both lists, in addition to the location of the pin, we keep the number of pins that corresponds to a given point. Before legalization in detailed placement, cell overlap can cause pins to have the same location.

Maintaining the number of real pins that corresponds to a point in a pointset (i.e., the multiplicity of that point) is necessary for efficient update of pin locations during placement. If a pin changes position during placement, the pointsets for the net connected to the pin must be updated. First, the original position of the pin must be removed from the movable point set. To remove the pin, one performs a binary search on the pointset. As multiple pins can have the same position, especially early in placement, without pointset the entire net would need to be traversed to see if any other pins share the same position as the pin that is moving. However, multiplicities make this information available in constant time. After the pin's location is found in the pointset, its multiplicity is reduced by 1. If this results in the position having a multiplicity of 0, the position is removed entirely. Insertion of the pin's new position is similar: first, a binary search is performed on the pointset. If the position is present, it's multiplicity is increased by 1. Otherwise, the position is added in sorted order with multiplicity 1.

Steiner weighted min-cut step by step. Pseudocode for minimizing Steiner wirelength in global placement is illustrated in Figure 5.4. At the beginning of min-cut placement, all movable cells are placed at the center of the first placement bin which encompasses the core area. Next, all the fixed and movable pointsets are initialized. To initialize

ALGORITHM 5.1: Minimizing Steiner wirelength in partitioning-based placement

```
    ▷ Input: queue of placement bins Q, netlist to place
    ▷ Output: placements of all the movable objects in netlist
 1  PLACEALLMOVABLESATTHECENTEROFTHETOPLEVELBIN()
 2  INITIALIZEMOVABLEPOINTSETS()
 3  INITIALIZEFIXEDPOINTSETS()
 4  while (EMPTY(Q) = FALSE)
 5      do bin ← DEQUEUE(Q)
 6         if (DETERMINEBINSIZE(bin) = SMALL)
 7           then CALLENDCASEPLACER(bin)
 8                REMOVENEWLYFIXEDPINSFROMMOVABLEPOINTSETS(bin)
 9                ADDNEWLYFIXEDPINSTOFIXEDPOINTSETS(bin)
10           else direction ← CHOOSECUTLINEDIRECTION(bin)
11                cutline ← CHOOSECUTLINEPOSITION(bin, direction)
12                (c1, c2) ← CALCULATECENTERSOFCHILDBINS(bin, cutline)
13                graph ← BUILDPARTITIONINGGRAPHWITHOUTNETS(bin, netlist)
14                foreach(net in netlist attached to a cell in bin)
15                    do terminals ← a list of terminal pin locations on net
                         by combining all points from net's fixed pointset
                         and points from net's movable pointset not contained within bin
16                       Calculate w1 using terminals, c1 and a Steiner evaluator
17                       Calculate w2 using terminals, c2 and a Steiner evaluator
18                       Calculate w12 using terminals, c1, c2 and a Steiner evaluator
19                       Adjust w1,w2,and w12 for consistency
20                       Add 0, 1 or 2 nets to graph whose weights and connectivity
                         are determined by w1,w2, and w12
21                childBins ← CALLPARTITIONER(bin, cutline, graph)
22                MOVECELLSTOCENTERSOFCHILDBINS(bin, childBins)
23                UPDATEPOINTSETSFORMOVEDCELLS(bin)
24                ENQUEUE(Q, childBins)
```

Figure 5.4: Minimizing StWL in top-down min-cut global placement.

a pointset, we sort it and change duplicates to multiplicities in a linear-time pass.

Before a partitioning instance is built for a bin, all nets that are incident to the bin must be examined in any min-cut placer. Usually any cell that is outside of the bin would be propagated to the border of the bin. We skip this step as this reduces the accuracy of the Steiner measurements. Instead we collect all the locations of terminals on this net. This includes all the fixed pins in addition to any movable pins that are outside of this bin. At

this step, other placers would check to see if the bounding box of terminals would contain the centers of the potential child bins (or would be checking for this condition while gathering the terminals on this net) and stop without adding this net to the partitioning problem. If this condition holds, the net is inessential to partitioning when optimizing for HPWL, but may not be inessential when optimizing for Steiner WL. Thus we cannot skip this net before calculating its three costs.

We calculate the three costs for each net by making calls to a particular Steiner evaluator. If the number of unique points that needs to be passed to the Steiner evaluator is larger than a certain threshold, we use HPWL evaluation instead purely for speed concerns. MST WL can be used for these large nets, but we have found routed wirelength degradation as compared to using HPWL (see Table 5.13). After making calls to the Steiner evaluator, we make checks to ensure consistency of the costs since the evaluators we are using are approximation algorithms for building RSMTs. For example we ensure that $w_1 \leq w_{12}$ by setting $w_1 = min(w_1, w_{12})$ and similarly for w_2. Also, we make sure that w_{12} is no larger than $min(w_1, w_2)+$ the rectilinear distance between the centers of the child bins. This is necessarily true because one has a tree that connects to all the terminals on the net and the center of partition 1, one can easily connect to the center of partition 2 with a single edge.

After constructing the partitioning instance with properly weighted nets, the partitioner runs and produces a solution. A cut-line is selected based on the partitioning (see Section 5.5 for more details), and new bins are constructed for the next cycle of min-cut placement to continue. When a new bin is constructed, cells that belong to that bin are placed at its center and all pointsets for nets incident to the bin must be updated. Since the pointset

Table 5.3: Runtime breakdown of global placement when minimizing StWL for ibm01-easy of the IBMv2 series of benchmarks [143]. "Partitioning problem construction" includes runtime for Steiner WL evaluators.

Global Placement Task	Runtime
Partitioning	53.56%
Partitioning problem construction	29.50%
End-case Placement	7.77%
Congestion Maps	6.44%
Pointset Maintenance	**0.86%**
Miscellaneous	1.87%

structures are sorted and have multiplicities, moving a pin to a new location takes time logarithmic in the number of pins on a net. Without multiplicities, the entire pointset would need to be rebuilt from scratch due to the removal of duplicates. Empirically, building and maintaining the pointset data structures takes less than 1% of the runtime of global placement, shown in Table 5.3. Pointsets must also be updated when bin is placed — movable pins get reassigned to the fixed-pin pointset. Note that partitioning only causes a movable pin to change position, and fixed pointsets are unaffected.

Performance. After implementing net-weighting based on pointsets, we compared three different Steiner evaluators to see their impact on runtime and solution quality. Based on the results discussed in the Section 5.6, we have chosen FastSteiner [78] for global placement, due to its reasonable runtime and consistent performance on large nets. Table 5.4 shows that the use of FastSteiner with our techniques lead to a reduction of StWL on IBMv2 benchmarks [143] by nearly 3% on average while using 36% additional runtime. Since min-cut placers are fast and extremely scalable, this is a very encouraging result.

The largest and smallest benchmarks (ibm01e and ibm12e) differ by 5x in size, but HPWL minimization consistently takes 75% of runtime for StWL minimization, suggest-

Table 5.4: Improving Steiner WL with FastSteiner [78]. Average HPWL, Steiner WL and placement runtimes are shown for the IBMv2 benchmarks [143]. Results are the average of five independent runs. All wirelengths are in meters. Optimizing StWL decreases StWL by 2.8%, increases runtime by 36% and HPWL by 1.4%.

Bench-mark	Minimizing HPWL			Minimizing Steiner WL		
	HPWL	StWL	Time (s)	HPWL	StWL	Time (s)
ibm01e	0.523	0.602	205	0.526	0.590	271
ibm01h	0.514	0.592	204	0.523	0.587	266
ibm02e	1.487	1.745	483	1.526	1.716	738
ibm02h	1.441	1.694	470	1.471	1.654	725
ibm07e	3.482	3.854	1134	3.484	3.747	1480
ibm07h	3.322	3.682	1092	3.401	3.659	1444
ibm08e	3.630	4.300	1484	3.757	4.241	2304
ibm08h	3.608	4.258	1446	3.646	4.131	2268
ibm09e	3.065	3.465	1207	3.130	3.408	1599
ibm09h	2.991	3.390	1179	3.037	3.313	1565
ibm10e	6.016	6.736	1918	6.088	6.619	2541
ibm10h	5.826	6.542	1885	5.830	6.356	2519
ibm11e	4.591	5.003	1740	4.608	4.888	2109
ibm11h	4.430	4.843	1679	4.478	4.757	2064
ibm12e	8.193	9.109	2235	8.321	8.990	3016
ibm12h	7.983	8.907	2215	7.966	8.621	2957
Ratio	1.000	1.000	1.000	1.014	0.972	1.364

ing that the ratio remains approximately constant regardless of the scale.

5.4 Detailed Placement Driven by Steiner Tree Length

Sliding-window optimizations for HPWL during detailed placement are quite common in modern placers. A recent technique of that variety models single-trunk Steiner trees and has had success in improving routability of FPGAs [73]. Unfortunately, it appears very slow. We have implemented two types of sliding-window optimizers directed at minimizing StWL using the FLUTE Steiner evaluator [44]. The first optimizer checks all possible linear orderings of small groups of cells and pieces of whitespace *exhaustively*. For the sake of efficiency, orderings of cells that are the same except for permutations of

whitespace pieces are only evaluated once. Other than this simple optimization, every cell ordering is generated and its StWL is calculated using FLUTE. The ordering with the least StWL is returned at the end of the procedure. Because of the exponential rate of growth of the number of permutations of n cells, namely $n!$, this exhaustive enumeration technique only scales to 4-5 cells.

The second optimizer also does linear placement, but uses a *dynamic programming* algorithm for an interleaving optimization similar in spirit to that presented by Jariwala and Lillis [73]. Given k cells, the algorithm splits the cells into groups A and B of sizes $n = k/2$ and $m = k-n$, respectively. The order of the cells in groups A and B is important and is the same as the initial configuration to the optimizer. The configurations that the algorithm examines are only those where cells in groups A and B are interleaved, but the relative order of cells from A and cells from B remain unchanged. For example, say we have the cells 1234abcd in this order. The ordering "1ab2cd34" is a legal ordering for the algorithm to consider, but the ordering "12a3bdc4" is not because c came before d in group B previously, but c is now behind d. The exact number of configurations that satisfy this interleaved ordering is $\frac{(n+m)!}{n!m!}$ which is much less than the $(n+m)! = k!$ possible configurations of the input.

First, the algorithm builds an n-by-m sized table of partial solutions. Entry (i, j) of the table contains the ordering with the best (smallest) StWL when interleaving the first i elements of group A and the first j elements of group B. The final answer is thus stored in position (n, m) of the table after the algorithm finishes. Table entries $(i, 0)$ and $(0, j)$ are trivial to calculate. The dynamic programming step of the algorithm computes entry (i, j)

Table 5.5: Statistics of the IBMv2 benchmarks [143].

Bench-mark	# Cells	# Nets	Whitespace		Metal layers
			easy	hard	
ibm01	12028	11753	14.88%	12.00%	4
ibm02	19062	18688	9.58%	4.72%	5
ibm07	44811	44681	10.05%	4.70%	5
ibm08	50672	48230	9.97%	4.84%	5
ibm09	51382	50678	9.76%	4.88%	5
ibm10	66762	64971	9.78%	4.92%	5
ibm11	68046	67422	9.89%	4.67%	5
ibm12	68735	68376	14.78%	9.94%	5

from entries $(i-1, j)$ and $(i, j-1)$. Element i of group A is added to the solution from entry $(i-1, j)$ and the StWL of the resulting placement is calculated from scratch with FLUTE. Similarly, element j of group B is added to the solution from entry $(i, j-1)$ and the StWL of this placement is calculated from scratch with FLUTE. The best of these two solutions in terms of StWL is taken to be the solution for entry (i, j). Calculating entries in row-major (or column-major) order will guarantee that all dependencies are satisfied.

Since the algorithm proceeds by filling in the table, the runtime of the algorithm is proportional to $n * m$ multiplied by the time to evaluate wirelength, while considering $\frac{(n+m)!}{n!m!}$ configurations. To speed up the process of evaluating wirelength, pointsets with multiplicities (see Section 5.3) are used in interleaving as well as exhaustive search. This dynamic programming approach has been shown to produce the optimal interleaving when HPWL is used for evaluation [73], but we have found that it does not necessarily produce min-StWL interleavings. On the other hand, it allows for windows of size 8-9 which is nearly twice that of exhaustive search.

Table 5.6 evaluates detailed placement on the IBMv2 benchmarks (statistics for which are presented in Table 5.5), with 4 cells per window during exhaustive enumeration and 8

Table 5.6: Detailed placement improves Steiner WL and routed WL. Average improvements and runtime (as a fraction of total placement time) from five independent runs are shown for the IBMv2 benchmarks [143].

Benchmark	Steiner WL improvement	Routed WL improvement	% Total runtime
ibm01e	1.047%	1.668%	11.66%
ibm01h	0.950%	4.046%	11.99%
ibm02e	0.735%	1.332%	10.89%
ibm02h	0.644%	0.363%	11.14%
ibm07e	0.647%	1.377%	11.51%
ibm07h	0.622%	3.288%	11.92%
ibm08e	0.553%	0.680%	11.27%
ibm08h	0.540%	1.620%	11.77%
ibm09e	0.716%	2.846%	13.00%
ibm09h	0.698%	3.041%	13.26%
ibm10e	0.662%	1.327%	12.42%
ibm10h	0.642%	0.225%	12.70%
ibm11e	0.639%	0.313%	11.65%
ibm11h	0.607%	0.273%	11.82%
ibm12e	0.682%	-0.789%	11.11%
ibm12h	0.619%	0.423%	11.50%
Average	0.688%	1.387%	11.83%

cells per window during interleaving. Such detailed placement alone reduces Steiner WL by 0.69% and routed WL by 1.4% while consuming 11.8% of total placement runtime.

5.5 Congestion-based Cut-line Shifting

In this section we introduce whitespace allocation based on congestion estimates during min-cut placement. This technique is essential to achieving routability, but in some cases increases routed wirelength, as seen in Figure 5.7.

One of the most important reasons that we use bisection instead of quadrisection is the flexibility that it allows in choosing the cut-line of a partitioned bin. Before partitioning, we first choose a direction for the cut-line, usually based upon the geometry of the bin. We then choose a tentative cut-line in that direction to split the bin roughly in half.

After the partitioner returns a solution, we have the flexibility to keep the cut-line as it was chosen before partitioning or to change it to optimize an objective. The WSA [93] technique, applied after placement, geometrically divides the placement area in half and estimates the congestion in both halves of the layout. It then allocates more area to the side with greater routing demand, i.e., shifts the cut-line, and proceeds recursively on the two halves of the design. In WSA, cells must be re-placed after the whitespace allocation. However, we can avoid this re-placement because our cells have not yet been placed and will be taken care of naturally during the min-cut process.

Cut-line shifting used to handle congestion necessitates a slicing floorplan. The only work in the literature that describes top-down congestion estimates and uses them in placement assumes a grid structure [16]. Therefore we develop the following technique: before each round of partitioning, we overlay the entire placement region on a grid. We choose the grid such that each placement bin is covered by 2-4 grid cells. We then build a congestion map using the last updated locations of all pins. We choose the mapping technique from [141] as it shows good correlation with routed congestion.

When cells are partitioned and their positions are changed, the congestion values for their nets are updated. Before cut-line shifting, the routing demands and supplies for either side of the cut-line are estimated with the congestion map. Given the bounding box of a region, we estimate its demand and supply by intersecting the bounding box with the grid cells of the congestion map. Grid cells that partially overlap with the given bounding box contribute only a portion of their demand and supply based on the ratio of the area of the overlap to the area of the grid cell. Using these, we shift the cut-line to equalize the ratio

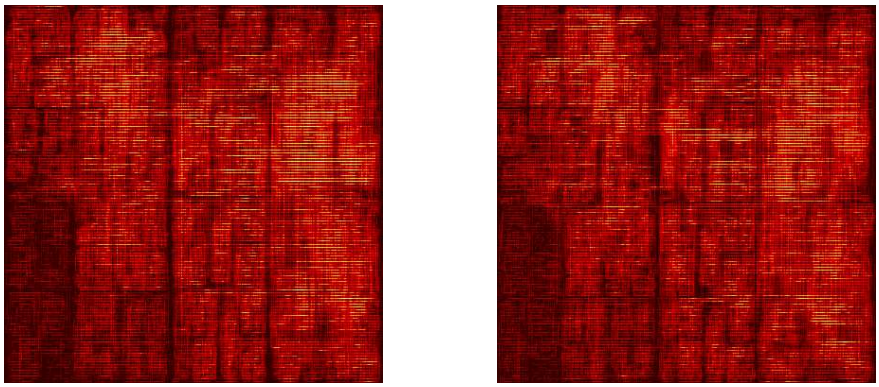


Figure 5.5: Congestion maps for the ibm01h benchmark: uniform whitespace allocation (produced with Capo -uniformWS) is illustrated on the left, congestion-driven allocation in ROOSTER is illustrated on the right. The peak congestion when using uniform whitespace is 50% greater than that for our technique. When routed with Cadence WarpRoute, uniform whitespace produces 3.95% overfull global routing cells and routes in just over 5 hours with 120 violations. ROOSTER's whitespace allocation produces 3.18% overfull global routing cells and routes in 22 minutes without violations.

of demand to supply on either side of the cut-line.

To show the effectiveness of this dynamic version of WSA, we plot congestion maps of placements of ibm01h produced with and without our technique in Figure 5.5. The left plot illustrates uniform whitespace allocation and the right plot congestion-driven whitespace allocation. Our whitespace allocation technique reduces maximum congestion by 50% and the number of overfull global routing cells from 3.95% to 3.18% (as reported by an industry router). We also post-process our placements with WSA and observe mixed results, as discussed below (see Table 5.10).

5.6 Steiner-tree Evaluators: Runtime, Accuracy and Fidelity

After implementing our technique to reduce StWL during global placement, we tested three different Steiner-tree evaluators to see how they would affect the runtime and solution quality of placement. The three evaluators used were Batched Iterated 1 Steiner (BI1ST)

[81], FastSteiner [78] and FLUTE [44]. We used each evaluator individually as well as combinations of all three. When using more than one evaluator at a time, we choose the smallest wirelength among all estimates since RSMT estimators overestimate actual RSMT length. Recall that FLUTE is known to be optimal for nets with nine or fewer pins and also much faster than other evaluators. Therefore, in mixed evaluators for nets with four to nine pins we use FLUTE exclusively.

Table 5.7 shows a runtime and solution quality comparison for all eight possible combinations of Steiner evaluator for the benchmark ibm01e. Runtimes and wirelengths are averages of five independent runs. The trends present for ibm01e are very similar for the other IBMv2 benchmarks. It is clear from the table that BI1ST gives the best solutions but uses the most runtime for a single evaluator. FastSteiner is very close to BI1ST in terms of solution quality, but uses much less runtime. Of the three pure evaluators, FLUTE is the least successful in terms of placement quality but is the fastest. We decided to use FastSteiner in global placement because it provided the best trade-off in terms of solution quality and runtime across all benchmarks.

Surprisingly, the mixed Steiner evaluators were outperformed by individual evaluators and hurt solution quality rather than improved it. This trend was even stronger on larger benchmarks. In particular, FastSteiner performed better than FastSteiner + FLUTE on ibm07. Certainly using the best of three Steiner evaluators makes estimates more accurate, but our global placement relies on *differences* between Steiner lengths rather than the *lengths themselves*. This suggests that the accuracy, measured by maximum error, of Steiner-tree estimation is not as important as its *fidelity*, which is defined as preserving

Table 5.7: Impact of Steiner evaluators during global placement (ibm01e). Total StWL and global placement runtime are listed for all combinations of three Steiner evaluators. In such combinations, the minimum Steiner length estimate is used in weighted partitioning.

Steiner evaluator(s)	Place time (s)	Steiner WL	Steiner WL Ratio
HPWL (no Steiner eval)	**141**	**0.5955**	**1.0000**
BI1ST + FastSteiner + FLUTE	202	0.5918	0.9937
BI1ST + FLUTE	186	0.5900	0.9907
BI1ST + FastSteiner	248	0.5893	0.9895
FLUTE	148	0.5886	0.9884
FLUTE + FastSteiner	158	0.5875	0.9866
FastSteiner	**180**	**0.5875**	**0.9866**
BI1ST	208	0.5861	0.9843

Table 5.8: A comparison of our work to best published routing results on the IBMv2 benchmarks [143]. All routed wirelengths (rWL) are in meters. A ratio greater than 1.0 indicates that our results are better on this benchmark suite. For all cases, ROOSTER outperforms best published routing results in terms of routed wirelength and via count. Published routing data for APlace 1.0 for ibm09-ibm12 is unavailable. Routing data for Capo 9.2, Dragon 3.01 and FengShui 2.6 were taken from [115] which did not list via counts. Routing uses a 24-hour time-out. Best legal rWL and via counts are in bold.

	ROOSTER			mPL-R + WSA [93]			APlace 1.0 /w cong [82]			Capo 9.2 [115]		Dragon 3.01 [143]		FengShui 2.6 [6]	
	rWL	#Vias	#Vio.	rWL	#Vias	#Vio.	rWL	#Vias	#Vio.	rWL	#Vio.	rWL	#Vio.	rWL	#Vio.
ibm01e	**0.733**	**122286**	0	0.77	127969	0	0.80	152489	0	0.779	0	0.843	0	time-out	932
ibm01h	**0.746**	**124307**	0	0.75	129648	0	0.75	150947	0	0.773	23	0.917	84	time-out	2698
ibm02e	2.059	**259188**	0	**1.89**	284396	0	2.05	299306	0	2.183	0	2.085	0	2.201	0
ibm02h	2.004	**262900**	0	**1.94**	296290	0	2.14	315786	0	2.080	0	2.216	0	2.277	0
ibm07e	**4.075**	**476814**	0	4.29	548765	0	4.18	559354	0	4.534	0	4.495	0	4.756	77
ibm07h	4.329	**489603**	0	4.43	579157	0	**4.29**	586129	1	4.591	0	4.523	0	4.707	251
ibm08e	**4.242**	**559636**	0	4.58	661733	0	4.58	681884	0	4.553	0	4.601	0	4.458	0
ibm08h	**4.262**	**574593**	0	4.49	684910	0	4.63	699411	0	4.768	0	4.961	0	5.056	52
ibm09e	**3.165**	**466283**	0	3.50	549568	0	-	-	-	3.357	0	3.705	0	3.520	0
ibm09h	**3.187**	**475791**	0	3.65	570032	0	-	-	-	3.336	0	3.494	0	3.395	0
ibm10e	**6.412**	**749731**	0	6.84	873311	0	-	-	-	6.591	0	6.948	0	6.809	0
ibm10h	6.602	**775018**	0	6.76	902026	0	-	-	-	**6.484**	0	6.982	0	6.716	0
ibm11e	**4.698**	**605807**	0	5.16	714824	0	-	-	-	5.039	0	5.371	0	5.301	0
ibm11h	**4.697**	**618173**	0	5.15	745015	0	-	-	-	4.941	0	5.400	0	5.260	0
ibm12e	**9.289**	**918363**	0	10.5	1127925	0	-	-	-	9.895	0	10.459	0	10.147	33
ibm12h	**9.289**	**938971**	0	10.1	1107551	0	-	-	-	10.145	0	9.904	0	time-out	3418
Ratio	1.000	1.000		1.055	1.156		1.042	1.119		1.056		1.107		1.093	

relative magnitudes between estimates.

5.7 Empirical Results

To test the quality of placements produced by ROOSTER, we ran it on the IBMv2 suite of benchmarks [143] and routed them using Cadence WarpRoute 2.4.41. All runs of

Table 5.9: A comparison of our work to the most recent version of mPL-R + WSA, APlace 2.04 and FengShui 5.1 on the IBMv2 benchmarks [143]. All routed wirelengths (rWL) are in meters. "Time" represents routing runtime in minutes. Note that while APlace 2.04 achieves overall smaller wirelength than ROOSTER, it routes with violations on 2 of the 16 benchmarks. Best legal rWL and via counts are in bold.

	ROOSTER				Latest mPL-R + WSA				APlace 2.04 -R 0.5				FengShui 5.1			
	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio.	Time
ibm01e	0.733	**122286**	0	42	**0.718**	123064	0	11	0.790	158646	85	132	0.804	166459	1630	1337
ibm01h	0.746	**124307**	0	32	**0.691**	213162	0	11	0.732	161717	2	121	0.807	166578	1451	1310
ibm02e	2.059	259188	0	13	**1.821**	**250527**	0	11	1.846	254713	0	9	2.324	383169	726	474
ibm02h	2.004	262900	0	14	**1.897**	**260455**	0	13	1.973	268259	0	14	2.284	343198	148	184
ibm07e	4.075	**476814**	0	17	4.130	492947	0	21	**3.975**	500574	0	17	4.387	591002	137	84
ibm07h	4.329	**489603**	0	19	4.240	516929	0	26	**4.141**	518089	0	23	4.632	617327	486	244
ibm08e	4.242	**559636**	0	17	4.372	579926	0	23	**3.956**	588331	0	18	5.050	740719	19	112
ibm08h	4.262	**574593**	0	20	4.280	599467	0	26	**3.960**	595528	0	18	4.759	725147	16	59
ibm09e	3.165	**466283**	0	11	3.319	488697	0	17	**3.095**	502455	0	11	3.462	517701	0	13
ibm09h	3.187	**475791**	0	11	3.454	502742	0	19	**3.102**	512764	0	12	3.348	510144	0	13
ibm10e	6.412	**749731**	0	22	6.553	777389	0	30	**6.178**	782942	0	23	6.599	807032	0	24
ibm10h	6.602	**775018**	0	27	6.474	799544	0	33	**6.169**	801605	0	28	6.661	812593	0	27
ibm11e	**4.698**	**605807**	0	15	4.917	633640	0	22	4.755	648044	0	18	5.419	671225	0	22
ibm11h	**4.697**	**618173**	0	16	4.912	660985	0	25	4.818	677455	0	24	5.452	679690	0	22
ibm12e	9.289	**918363**	0	36	10.185	995921	0	57	**8.599**	921454	0	32	9.829	1172981	6	73
ibm12h	9.289	**938971**	0	43	9.724	976993	0	50	**8.814**	961296	0	50	10.333	1344067	466	448
Ratio	1.000	1.000			1.007	1.069			0.968	1.073			1.097	1.230		

placement and routing were performed on 3.2GHz Intel Pentium 4 processors with 1GB of RAM. All runs of randomized placers, including ROOSTER, are the average results for the best of three independent placements (only the best of the three independent placements is routed and the results of three such sets of placements are averaged). Statistics for the IBMv2 benchmarks are shown in Table 5.5. The effectiveness of each of the approaches that make up ROOSTER is depicted in Figure 5.7. A comparison of ROOSTER against the best published results for several competitive placers is shown in Table 5.8. A ratio greater than 1.0 indicates that our results are better for this benchmark suite, which is true for all the routed wirelengths and via counts of previously published results.

Most of the placers whose best published results are shown in Table 5.8 have more recent binaries which we evaluate in Table 5.9. We ran Dragon 4.0 in fixed-die mode, but it consistently crashed and we are unable to show results for it. Table 5.9 shows that the latest version of mPL-R + WSA has slightly worse rWL (0.7%) when compared to ROOSTER and 6.9% higher via count. Congestion-driven APlace 2.04 (using congestion

Table 5.10: Results when applying various post-processors to our placements for the IBMv2 benchmarks [143]. All routed wirelengths (rWL) are in meters. "Time" represents routing runtime in minutes. WSA shows improvement on some of our placements, but increases routed wirelength and via counts on the largest benchmarks. The detailed placers of Dragon 4.0 and FengShui 5.1 decrease the routability of our placements by increasing rWL and via count on all benchmarks and the addition of violations. Best legal rWL and via counts are in bold.

	ROOSTER				ROOSTER + WSA				ROOSTER + Dragon 4.0 DP				ROOSTER + FengShui 5.1 DP			
	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio.	Time
ibm01e	0.733	**122286**	0	42	**0.718**	122873	0	7	0.790	133498	0	92	0.850	162248	155	152
ibm01h	0.746	124307	0	32	**0.725**	**124063**	0	10	0.800	176562	36	166	0.858	176585	257	265
ibm02e	2.059	259188	0	13	**2.000**	**256155**	0	10	2.164	278854	0	19	2.215	347022	129	77
ibm02h	2.004	262900	0	14	**1.978**	**262022**	0	11	2.004	271237	0	33	2.234	345638	285	171
ibm07e	4.075	476814	0	17	**3.953**	**470104**	0	13	4.175	502808	0	19	4.498	581269	563	44
ibm07h	4.329	489603	0	19	**4.091**	**489067**	0	19	4.721	593629	76	21	4.885	617061	870	86
ibm08e	4.242	559636	0	17	**4.231**	**559010**	0	16	4.443	598266	0	18	4.662	684313	276	27
ibm08h	4.262	**574593**	0	20	**4.240**	577879	0	19	4.491	619733	0	36	4.794	714798	768	207
ibm09e	**3.165**	**466283**	0	11	3.200	473605	0	11	3.392	502967	0	11	3.718	573996	583	22
ibm09h	**3.187**	**475791**	0	11	3.205	480961	0	11	3.328	511174	0	12	3.688	587486	630	19
ibm10e	**6.412**	**749731**	0	22	6.420	755673	0	21	6.759	798405	0	23	7.214	905508	229	18
ibm10h	6.602	**775018**	0	27	**6.544**	781897	0	26	6.523	804478	0	29	6.943	911878	296	34
ibm11e	**4.698**	**605807**	0	15	4.746	613437	0	15	4.879	644060	0	15	5.308	735762	492	59
ibm11h	**4.697**	**618173**	0	16	4.716	625654	0	16	4.830	654948	0	16	5.288	755418	591	77
ibm12e	**9.289**	**918363**	0	36	9.333	930397	0	30	9.427	953405	0	39	9.888	1087932	10	44
ibm12h	9.289	**938971**	0	43	**9.282**	942551	0	39	9.260	966280	0	47	9.786	1102197	312	66
Ratio	1.000	1.000			0.990	1.004			1.041	1.089			1.114	1.248		

parameter 0.5) has rWL 3.24% smaller than ours, but 7.32% more vias and violations on 2 of the 16 benchmarks.

Since our cut-line shifting for congestion can be viewed as a dynamic version of the WSA post-processing technique, we were interested in seeing how WSA or other detailed placement techniques would affect the routability of our placements. Table 5.10 shows that WSA is able to improve our wirelength by 1.0% with a 0.4% increase in via count. Direct comparisons show that the most improvement is obtained on the ibm01 and ibm02 benchmarks. In contrast, the detailed placers of Dragon 4.0 and FengShui 5.1 degrade the routability of our placements, increasing routed wirelength, via counts and violations.

The Faraday series of five mixed-size benchmarks with routing information is derived from circuits released by the Faraday Corporation [1]. To see if ROOSTER techniques are applicable when fixed obstacles are present, we fixed the movable macros in the design (as shown in Figure 5.6) and used the resulting benchmarks with ROOSTER. All benchmarks

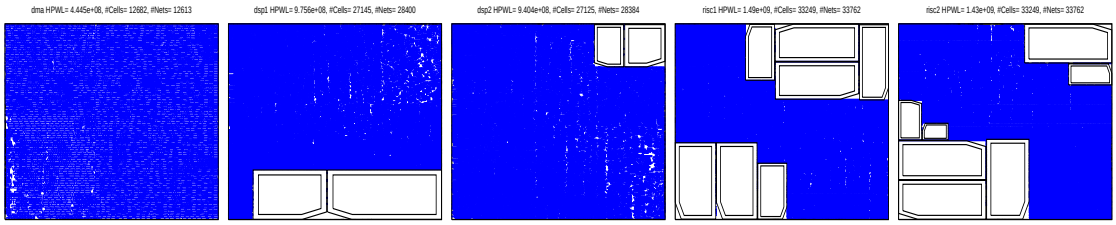
Table 5.11: A comparison of ROOSTER to Cadence AmoebaPlace on the IWLS 2005 Benchmarks [72]. All routed wirelengths (rWL) are in meters. "Time" represents routing runtime in minutes. ROOSTER is outperforms AmoebaPlace by 12.0% in rWL and 1.1% in via counts (without orientation constraints the improvements are 26.5% and 3.2%, respectively). Best rWL and via counts are in bold.

Benchmark	ROOSTER + NanoRoute				ROOSTER (w/o row orient) + NanoRoute				AmoebaPlace + NanoRoute			
	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio.	Time
aes_core	1.339	**125939**	2	32	**1.271**	126645	1	50	1.657	131049	1	28
ethernet	7.287	467777	1	27	**6.145**	**413323**	2	257	7.745	471800	1	28
mem_ctrl	1.061	**87276**	0	22	**0.890**	89153	0	33	1.224	90067	0	21
pci_bridge32	1.336	**114880**	0	35	**1.176**	115675	0	59	1.598	117326	2	35
usb_funct	0.995	**84717**	0	19	**0.860**	85329	0	33	1.106	85739	0	19
vga_lcd	25.906	1131591	2	57	**24.447**	1083504	1	173	25.405	**1076178**	2	90
Ratio	1.000	1.000			0.885	0.979			1.120	1.011		

were routed using Cadence WarpRoute 2.4.41. A comparison of ROOSTER placements to the original placements of the benchmarks produced by Silicon Ensemble Ultra v5.4.126 (details on the construction of the benchmarks can be found in [1, Appendix A]) are shown in Table 5.12. Results for APlace 2.04 and mPL-R are not shown as they crashed on all but the DMA benchmark (the only Faraday benchmark without macros). Compared to the SEUltra placements, ROOSTER improves routed WL by 11.2% and via counts by 4.8%.

Previous work has compared mPL-R/WSA and APlace with Cadence QPlace and found mPL-R/WSA to have the best results on IBMv2 benchmarks [93]. Since we show better results than mPL-R/WSA, ROOSTER should also compare favorably with QPlace on the IBMv2 benchmarks. Capo has demonstrated comparable performance to QPlace on another set of industry benchmarks [21]. Since ROOSTER considerably improves upon Capo, we expect similar improvements over QPlace.

We also performed placement experiments on the IWLS 2005 benchmarks [72]. Unlike the IBMv2 benchmarks which use a 0.25 μm cell library, the IWLS 2005 benchmarks use a Cadence 0.18 μm library. Table 5.11 compares ROOSTER with Cadence AmoebaPlace from SOC Encounter 4.1 on a few of the IWLS 2005 designs. All of the bench-



(a) DMA (b) DSP1 (c) DSP2 (d) RISC1 (e) RISC2

Figure 5.6: The ICCAD'04-Faraday benchmarks placed by ROOSTER. Macros are depicted with double outlines and are fixed.

Table 5.12: Routing results on the Faraday benchmarks with movable macro blocks fixed [1]. All routed wirelengths (rWL) are in meters. "Time" represents routing runtime in minutes. Best rWL and via counts are highlighted.

Bench-	ROOSTER				Silicon Ensemble Ultra v5.4.126			
mark	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio	Time
DMA	**0.554**	**116414**	0	3	0.644	125328	0	3
DSP1	**1.110**	209274	0	5	1.224	**204863**	0	6
DSP2	**1.067**	**194971**	0	6	1.230	207521	0	6
RISC1	**1.868**	**328699**	5	9	1.957	345615	4	6
RISC2	**1.786**	**324278**	5	7	1.959	347515	2	5
Ratio	1.000	1.000			1.112	1.048		

marks were routed with Cadence NanoRoute. The two sets of results for ROOSTER differ in how they handle cell orientations in rows that have nontrivial orientations. A full discussion on the orientations of standard cells and pin access is beyond the scope of this work, but the version of ROOSTER that does not respect nontrivial row orientations takes much longer to route than the version that does but can achieve significantly smaller routed wirelengths. ROOSTER improves upon AmoebaPlace in rWL by 12.0% and 1.1% in via count. This empirical comparison to a placement tool from Cadence also suggests that our techniques are superior to those published by Cadence in 1994 [39]. We did not have success using APlace 2.04 and mPL-R on these designs. APlace 2.04 completed global placement on all but the largest benchmark, but terminated with an error message during legalization. mPL-R crashed on all of the benchmarks that were tried.

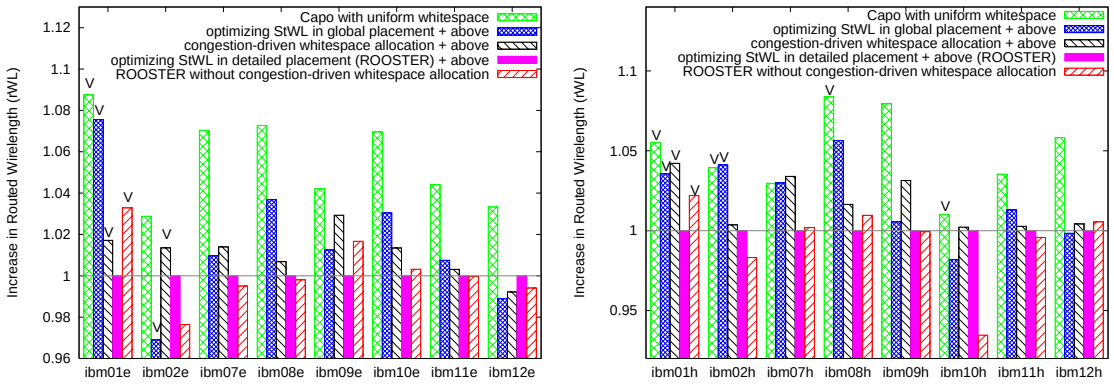


Figure 5.7: Impact of individual optimizations on the rWL produced by ROOSTER. "V" indicates violations in routing.

To see if the routed wirelength of ROOSTER placements could be improved without dramatically increasing its runtime, we attempted to add Minimal Spanning Tree (MST) wirelength into the ROOSTER framework. Recall that if a net has more than a certain threshold of pins, 20 for our experiments, ROOSTER uses HPWL to evaluate the net instead of a Steiner evaluator for reasons of speed. As MST wirelength is a more accurate estimator of routed wirelength than HPWL and is faster to calculate than StWL, we replaced HPWL with MST wirelength for large nets when calculating weights for partitioning.

Results of adding MST into ROOSTER are shown in Table 5.13. As we can see, using MST in place of HPWL in ROOSTER increases rWL by 0.7% and via count by 5.1% while reducing routability as 6 benchmarks have violations. Since the fidelity of wirelength evaluator is crucial, we performed an additional experiment where all net weights were calculated using MST WL. Table 5.13 that this increases rWL by 1.5% and via count by 5.0% and reduces routability on 7 benchmarks. These results reinforce our hypothesis that Steiner WL is a better placement optimization objective than MST wirelength.

Table 5.13: The impact of replacing HPWL (for high degree nets) and StWL (for all nets) with MST as the wirelength evaluator for ROOSTER on the IBMv2 benchmarks. All routed wirelengths (rWL) are in meters. "Time" represents routing runtime in minutes. The ratios are with respect to ROOSTER's performance described in Table 5.8. Legal improvements to ROOSTER in rWL and via counts are highlighted in bold.

Bench-mark	HPWL replaced by MST				StWL replaced by MST			
	rWL	#Vias	#Vio.	Time	rWL	#Vias	#Vio.	Time
ibm01e	0.768	149073	40	188	0.754	136724	2	54
ibm01h	0.768	161339	121	231	0.764	157896	32	184
ibm02e	2.017	281313	2	18	**2.012**	**254610**	0	16
ibm02h	2.010	288491	9	48	2.185	312547	119	89
ibm07e	4.105	481189	0	26	4.102	**475751**	0	26
ibm07h	4.410	528926	18	44	4.214	527378	20	63
ibm08e	4.327	564834	0	28	4.301	**559318**	0	27
ibm08h	4.328	580717	0	33	4.395	618671	4	34
ibm09e	3.192	470294	0	17	3.267	470715	0	18
ibm09h	**3.150**	**475043**	0	18	3.230	478005	0	19
ibm10e	**6.283**	**746000**	0	32	6.538	794192	1	36
ibm10h	6.577	766170	0	38	**6.559**	**765255**	0	37
ibm11e	4.784	608935	0	25	4.798	608887	0	24
ibm11h	4.719	620048	0	24	4.750	619988	0	25
ibm12e	9.277	926201	0	64	9.347	916887	0	55
ibm12h	9.267	991382	1	57	9.301	980202	1	52
Ratio	1.007	1.051			1.015	1.050		

5.8 Conclusions and Further Work

We have presented techniques which leverage recent advances in RSMT construction [44, 78] to optimize Steiner wirelength in global and detailed placement with only a modest increase in runtime, which are currently usable only in our placement algorithm ROOSTER which is freely available as part of the UMpack (`http://vlsicad.eecs.umich.edu/BK/PDtools/`). As the results of Figure 5.7 show, the optimization of Steiner tree lengths in global placement is the main source of improved wirelength. However, whitespace distribution is critical to prevent routing violations, even at the cost of increased wirelength. ROOSTER outperforms best published routed wirelength results

for Dragon, Capo, FengShui, mPL-R/WSA and APlace by 10.7%, 5.6%, 9.3%, 5.5% and 4.2% respectively. Via counts, especially important at 90nm and below, are improved by 15.6% over mPL-R/WSA and 11.9% over APlace. Further improvements by others in Steiner-tree construction and congestion maps can make our results better. In particular, if FLUTE becomes faster and can process large nets with high fidelity, detailed placement windows can increase.

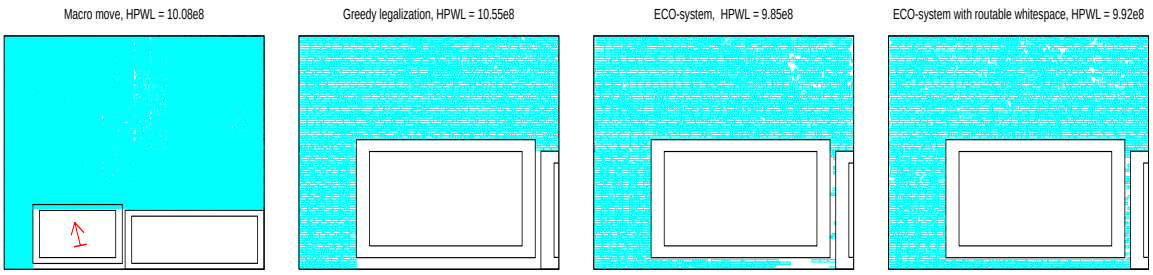
Properly accounting for obstacles in placement is an area that could benefit significantly from our StWL minimization techniques. An obstacle-aware Steiner evaluator could be used directly in our implementation for nontrivial improvement. In addition to handling blockages, both Steiner-tree evaluators used in ROOSTER (FLUTE [44] and FastSteiner [78]) can handle arbitrary per unit-costs of horizontal and vertical wires. This may provide a safer means of balancing the demand for horizontal and vertical routing resources (similarly motivated cut-line selection did not improve results in our tests).

Our technique may conceivably be extended to improve circuit timing — this requires the ability to estimate the per-net timing differential based on Steiner trees which we already compute. Extensions to optimize timing may require block-based static timing analysis. Even more accessible would be a similar extension to optimize dynamic power. In particular, in designs with multiple clock domains, we could optimize clock trees during global placement by estimating the lengths of bounded-skew clock trees using algorithms such as BST-DME.

CHAPTER VI

Incremental Placement and Applications to Physical Synthesis

In his keynote talk at ISPD 2006, Cadence CTO Ted Vucurevich expressed the need for "re-entrant, heterogeneous, incremental, and hierarchical" tools for EDA to handle the challenges of next-generation designs [61]. However, the importance of this problem has been realized much earlier, as Cong and Sarrafzadeh surveyed the state-of-the-art in incremental physical design techniques in 2000 and found these techniques to be largely "unfocused and incomplete" [51]. Kahng and Mantik also found disconnects between the relative strengths of incremental optimizers and perturbation techniques [76]. They conclude that CAD tools of the time "may not be correctly designed for ECO-dominated design processes" [76]. Recent work by Kahng and Reda suggests that certain types of netlist transformations are not handled well by re-placement from scratch, which also motivates incremental tools [80]. Considerable progress has been made since 2000, e.g., in incremental placement [2,5,18,52,65,69,92,93,95,96,100,113], but there is no consensus on the main tasks solved by incremental tools and how these tasks should be solved. While incremental physical design is not new, it remains a difficult, high-value goal.



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

Figure 6.1: Legalization of a macro move in the ICCAD'04-Faraday design DSP1 [1]. In (a), the left-most macro is moved toward the north-west corner of the design. This move causes overlap with standard cells and also areas of empty space below and to the right of the macro. The remaining three images are zoomed-in legal placements of this design. In (b), a greedy algorithm which tries to minimize cell movement is applied. Overlap is removed, but the empty space below and to the right of the macro remain unutilized which can be detrimental to routability. (c) shows the placement as legalized by our tool ECO-system. ECO-system improves wirelength and makes use of much of the area vacated by the macro. Lastly, (d) shows how ECO-system can distribute cells and whitespace so as to ensure routability and/or satisfy minimum whitespace constraints.

We focus on incremental placement legalization and improvement in large-scale layout. The need for such legalization typically arises in two contexts. The first is the separation of placement into global and detail, where rough placements are produced first and incrementally improved to avoid overlaps and fit into cell sites. This is common for analytical placers (APlace [82], mPL [29]) that approximate site constraints, while partitioning-driven tools (Capo [115], PolarBear [50]) and annealing-based tools (mPG [31], Parquet [3]) adopt correct-by-construction frameworks requiring little post-processing.

However, the second context for legalization appears entirely unavoidable. During physical synthesis, timing-critical gates may be powered up and other gates may be powered down. These changes affect gate size and typically create overlaps [92]. Buffer insertion often leads to similar area violations, which must be resolved by legalization.

The success of such legalization depends on how much the areas have changed, in what patterns, and the strength of a given legalizer. In particular, the legalization of mixed-size and block-based designs with obstacles remains very challenging [107].

Our work is focused on the design of a powerful and robust ECO tool that applies adequate amounts of replacement, in the right locations, to accommodate necessary design changes. To be useful in high-level and physical synthesis, such a tool must be able to entirely replace sections of the netlist, e.g., logic added to the design.

While practical considerations call for an interaction between global placers and legalizers, traditional work on ECO and detailed placement focuses on stand-alone tools incapable of global placement. An attractive, but yet unexplored solution would be to extend an existing global placer to an incremental mode where it would automatically identify layout regions and sections of the netlist that need repair, but preserve satisfactory regions. In this work, we propose such an extension, identify and develop new components that allow a global placer to act like a powerful ECO tool, and develop a competitive implementation based on the open-source Capo tool.

As this tool can always resort to calling global placement on the entire design, it robustly handles a full range of modern designs, including those with obstacles and movable macros. Time-consuming global placement is not used when the initial placement is good.

We formulate the basic requirements for ECO placement and offer relevant algorithms. Our tool, ECO-system, is many times faster than a global placer and increases wirelength only slightly. ECO-system outperforms APlace's native legalizer on APlace global placements by over 1% in HPWL while running four times faster. ECO-system supports ex-

Table 6.1: A comparison of several legalization and incremental placement techniques. For each of the techniques, its compatibility with fixed objects or macros as well as what general techniques it uses are listed. ECO-system is compared with XDP [52] in Section 6.5. (†) Support of the feature by this technique is unclear. See Section 6.1 for more details. (‡) Recent versions of Capo, the basis of ECO-system, use linear programming and network flows in detailed placement, but they are beyond the scope of this work.

		Capo [115]	Diffusion [96, 113]	DOMINO [56]	WSA [92, 93]	XDP [52]	**ECO-system**
Features	Fixed-object support	✓	✓			†	✓
	Macro support	✓	†		†	✓	✓
	Whitespace redistribution				✓		✓
Techniques used	Cell swapping					✓	✓
	Greedy legalization	✓	✓		✓	✓	✓
	Linear programming					✓	‡
	Network flows			✓		✓	‡
	Sliding-window optimization				✓	✓	✓

tensive cell resizing producing legal results that mirror the original with virtually the same HPWL while having minimal impact on timing. Unlike WSA [92,93], we handle obstacles and displace cells an order of magnitude less.

The rest of this chapter is structured as follows. In Section 6.1 we review previous work. Key requirements and a likely interface are discussed in Section 6.2. We present ECO-system in Section 6.3. Support for high-level and physical synthesis is discussed in Section 6.4. In Section 6.5 we show empirical results and conclude in Section 6.6.

6.1 Previous Work in Incremental Placement

Previous work on legalization, incremental placement and detailed placement can be broken into three fairly distinct stages: (i) cell spreading, (ii) legalization through sim-

ple end-case techniques, and (iii) refinement of the legalized placement. For the first stage, several algorithmic paradigms have been applied in the literature such as network flows [18,52,56,95], linear programming [52], top-down whitespace injection [92,93] and diffusion gradients [113]. For end-case legalization, generally placers use greedy movement of cells such as in Capo [115], the Tetris legalizer [65] in FengShui [6], and greedy packing in DOMINO [56]. After legalization, placement refinement is done in sliding windows of one or more rows using optimal end-case placers based on branch-and-bound [23] or dynamic programming [69], as well as cell swapping [111].

One major theme in much of the literature is minimizing the total movement of cells in the design during legalization [18]. While our legalizer achieves remarkably small total/average movement, we point out that in general this does not always lead to minimal increase in interconnect parameters as shown in [11]. A legalization with minimal total cell displacement may cause a few cells to move a great distance. Better timing may be achieved by legalization with greater average movement, and even if the average movement is the same, there can be many alternative replacements.

Cell spreading. The term "cell spreading" has been used by several authors in different contexts. In particular, several papers describe algorithms that do not take interconnect into account, while ECO-system includes interconnect optimization. Some of these publications do not describe the handling of movable and especially fixed obstacles, while ECO-system handles both, as confirmed by our experiments.

DOMINO [56] legalizes by splitting cells into pieces of identical sizes, solving a flow formulation to minimize movement, and finally reassembling the cell pieces. This lim-

its the effectiveness of DOMINO to cells of similar sizes. Existing implementations of DOMINO do not account for obstacles and shift all cells to the left, limiting their applicability to modern placement instances, such as those from the ISPD'05 contest [104]. Flow-based legalization methods such as those used in [18, 95] divide the core area into regions and redistribute cells between neighboring regions until no region has more cell area than available site area. These techniques can handle movable macros by fixing them early in the legalization process.

In [92, 93] cells are incrementally placed by injecting whitespace in a top-down fashion. The placement region is divided into a regular grid with geometric bisection steps (based only on the size and shape of the region, not taking into account the cells, macros or fixed obstacles therein), and whitespace is injected based on some particular objective (routing congestion in [93], gate sizing and buffer insertion in [92]). Whitespace injection is done by shifting the geometric cut-lines to change the whitespace balance in regions. When cut-lines are shifted, the positions of the cells in the affected regions are scaled. Whitespace injection can cause significant overlap due to scaling, especially in the presence of fixed obstacles or movable macros as in the ISPD 2005 Contest benchmarks [104]. To remove these overlaps, a standard legalization step must be applied followed by window-based detailed placement to recover HPWL. It is unclear how well this technique may work on difficult block-packing instances [107]. In addition, the most current implementation of this technique, WSA, does not support macros. The technique may also fail in cases of extreme overlap, such as global placement by analytical placers, as large areas of the placement will be essentially random. The authors of [92] report an

average displacement of 2.1% of the core half-perimeter per cell, whereas the displacements observed with our technique are an order of magnitude smaller.

The diffusion technique of [113] legalizes by dividing the core area into a regular grid. Cells move from areas of high congestion to lower congestion (moving around fixed obstacles) and their directions and speeds are determined by solving equations similar to those in the process of chemical diffusion [113]. New placements are generated at each time step of the diffusion and the first solution which satisfies area constraints is taken to minimize runtime and cell movement [113]. End-case legalizers work within the grid regions to produce a final legal placement, but this may be impaired by difficult block-packing instances [107]. The work in [96] improves that in [113], but does not measure its impact on wirelength, congestion or timing.

The XDP technique [52] uses a combination of constraint graphs, network flows, linear programming and greedy cell movement for legalization of mixed-size designs. Overlaps between macros are legalized first by building constraint graphs until all macros can legally fit into the core. After the constraint graph is finalized, a linear programming instance is built and solved to remove macro overlap and move macros minimally. It is unclear if this technique supports fixed macros. Standard cells are legalized with a greedy heuristic similar to that of FengShui [6], with the addition of flow-based methods [18, 95] as necessary. After legalization, window-based detailed placement techniques are used to improve HPWL. We evaluate XDP in our experiments.

Greedy legalization. FengShui [6] uses a simple packing algorithm by Hill [65] that is reminiscent of the Tetris game. Such legalization fares poorly in designs with large

amounts of whitespace, as shown by the results of the ISPD 2005 Placement Contest. Capo uses two greedy legalizers for its global placements: one for macros and another for standard cells [115]. The macro overlap legalizer tries to move macros as little as possible so as not to affect neighboring standard cells. If space is available, standard cells are legalized via shifting. Otherwise cells are swapped between rows greedily until no row is overfull. Fixed obstacles are handled implicitly as they fracture rows [115].

Macro legalization. It was shown that a fixed-outline floorplanner based on Simulated Annealing with sequence pairs could be used to remove overlap [2]. Techniques in [136] improve on [2] and show how to legalize with minimal perturbation. Removal of overlap between macros can be especially difficult given hard instances of block-packing [107]. To handle such instances, the authors of [107] modify B*-trees to account for obstacles. The recent FLOORIST tool [100] uses constraint satisfaction to remove macro overlap.

6.2 Requirements of Incremental Placement

Design optimizations that require incremental placement can alter a design in many ways [57] such as (see also Section 6.4):

- Changing cell dimensions or net weights/criticalities
- Adding/Removing various constraints, such as density (to promote routability), regions (to address timing), etc.
- Inserting/Removing cells (with or w/o initial locations), nets or macros
- Adding/Moving obstacles (memories, IP blocks, RTL macros, etc.)

Generally these transformations create illegality in localized regions of a design and/or create opportunities for improving an existing placement. All of these transformations can

be dealt with by performing placement from scratch, but this is undesirable: (i) replacement can be slow, (ii) the transformations may assume that they are applied to the current layout, and placement from scratch may invalidate them, and (iii) the current layout may include *intangibles* such as designer intent, or be optimized for novel objectives not accounted for by the placement tool. Cong and Sarrafzadeh point out that incremental placers need to be able to trade off potentially several design objectives when operating on a placement [51].

In addition to preserving the original placement, a legalizer must also be able to completely replace sections of the placement that are deemed too suboptimal after design alterations. For example, if all of the cells are moved on top of one another at the center of the placement area, the legalizer should have the ability to replace all of the cells as the initial placement gives little useful information about a legal placement of the design. While this example is not typical of legalization as a whole, it is quite possibly the case for small sections of an illegal placement. This pathological case is not considered by most legalization techniques (such as those described in Section 6.1).

Take for example the case when new cells are added to a design. If the new cells are added to isolated regions of the design, such as during buffer insertion, traditional techniques that perturb the design only slightly are most likely appropriate. Yet, timing optimization may call for pipelining of a multiplier or changing an adder to a different type. Adding a significant amount of new logic to an already placed and optimized design will require the functionality of a full-blown placer rather than just cell spreading to avoid degrading the design's wirelength and timing characteristics.

ALGORITHM 6.1: Top-down ECO placement

```
    ▷ Input: queue of placement bins Q, netlist to place
    ▷ Output: placements of all the movable objects in netlist
 1  while (EMPTY(Q) = FALSE)
 2      do bin ← DEQUEUE(Q)
 3         if (BINMARKEDTOPLACEFROMSCRATCH(bin) = FALSE)
 4           then if (BINOVERFULL(bin) = TRUE)
 5                  then MARKBINTOPLACEFROMSCRATCH(bin)
 6                       break
 7                direction ← CHOOSECUTLINEDIRECTION(bin)
 8                cutline ← cut-line which has the smallest net-cut
                  considering cell area balance constraints
 9                if (cutline causes overfull child bin)
10                  then MARKBINTOPLACEFROMSCRATCH(bin)
11                       break
12                partitioning ← INDUCEPARTITIONING(bin, cutline)
13                netcut ← EVALUATENETCUT(partitioning)
14                newPartitioning ← RUNFLATFIDDUCIAMATTHEYSES(partitioning)
15                newNetcut ← EVALUATENETCUT(newPartitioning)
16                if (newNetcut/netcut < IMPROVEMENTTHRESHOLD)
17                  then MARKBINTOPLACEFROMSCRATCH(bin)
18                       break
19                childBins ← CREATECHILDBINS(bin, partitioning, cutline)
20                ENQUEUE(Q, childBins)
21         if (BINMARKEDTOPLACEFROMSCRATCH(bin) = TRUE)
22           then if (DETERMINEBINSIZE(bin) = SMALL)
23                  then CALLENDCASEPLACER(bin)
24                  else direction ← CHOOSECUTLINEDIRECTION(bin)
25                       cutline ← CHOOSECUTLINEPOSITION(bin, direction)
26                       graph ← BUILDPARTITIONINGGRAPH(bin, cutline, netlist)
27                       childBins ← CALLPARTITIONER(bin, cutline, graph)
28                       MARKBINSTOPLACEFROMSCRATCH(childBins)
29                       ENQUEUE(Q, childBins)
```

Figure 6.2: Our top-down partitioning-based ECO placement algorithm. Lines 3-21 and 28 differ from traditional min-cut placement.

6.3 Top-down Legalization

To develop a strong ECO tool, we build upon an existing global placement framework and must choose between analytical and top-down. The main considerations include robustness, the handling of movable macros and fixed obstacles, as well as consistent routability of placements and the handling of density constraints. Based on recent empirical evidence [107, 116, 120], the top-down framework appears a somewhat better choice.

Indeed the 2 out of 9 contestants in the ISPD 2006 Competition that satisfied density constraints were top-down placers. However, analytical algorithms can also be integrated into our ECO-system when particularly extensive changes are required. We base ECO-system on the open-source min-cut placer Capo [115].

6.3.1 General Framework

The goal of ECO-system is to reconstruct the internal state of a min-cut placer that could have produced a given placement **without the expense of global placement**. Given this state, we can choose to accept or reject previous decisions based on our own criteria and build a new placement for the design. If many of the decisions of the placer were good, we can achieve a considerable runtime savings. If many of the decisions are determined to be bad, we can do no worse in terms of solution quality than placement from scratch. After this modified global placement, we use a subset of Capo's detailed placement to guarantee legality. An overview of the application of ECO-system to an illegal placement is depicted in Figure 6.3. See the algorithm in Figure 6.2.

To rebuild the state of a min-cut placer, we must reconstruct a series of cut-lines and partitioning solutions efficiently. To extract a cut-line and partitioning solution from a given placement bin, we examine all possible cut-lines as well as the partitions they induce. We start at one edge of the placement bin (left edge for a vertical cut and bottom edge for a horizontal cut) and move towards the opposite edge. For each potential cut-line encountered, we maintain the cell area on either side of the cut-line, the partition induced by the cut-line and the net cut.

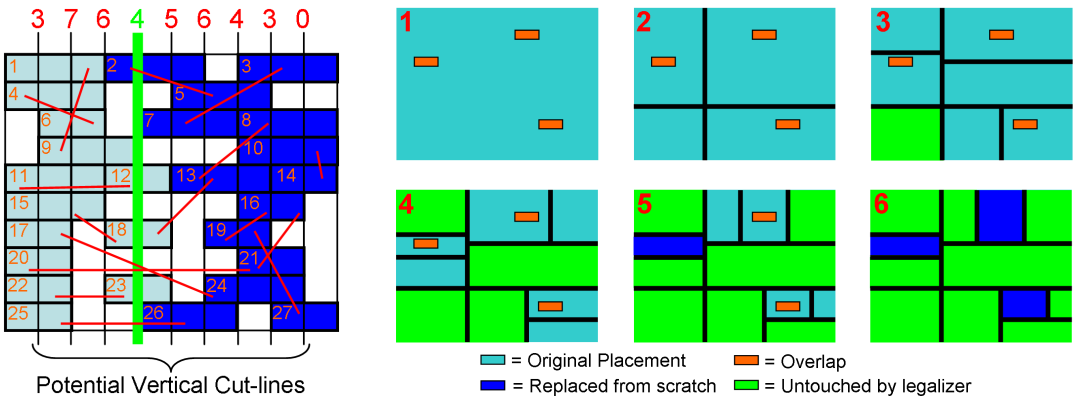


Figure 6.3: Fast legalization by ECO-system. The image on the left illustrates choosing a vertical cut-line from an existing placement. Nets are illustrated as red lines. Cells are individually numbered and take 2 or 3 sites each. Cut-lines are evaluated by a left-to-right sweep (net cuts are shown above each line). A cut-line that satisfies partitioning tolerances and minimizes cut is found (thick green line). Cells are assigned to "left" and "right" according to the center locations. On the right, placement bins are subdivided using derived cut-lines until (i) a bin contains no overlap and is ignored for the remainder of the legalization process or, (ii) the placement in the bin is considered too poor to be kept and is replaced from scratch using min-cut or analytical techniques.

6.3.2 Fast Cut-line Selection

For simplicity, assume that we are making a vertical cut and are moving the cut-line from the left to the right edge of the placement bin (the techniques necessary for a horizontal cut are analogous). Pseudocode for choosing the cut-line is shown in Figure 6.4. To find the net cut for each possible cut-line efficiently, we first calculate the bounding box of each net contained in the placement bin from the original placement. We create two lists with the left and right x-coordinates of the bounding boxes of the nets and sort them in increasing x-order. While sliding the cut-line from left to right (in the direction of increasing x-coordinates), we incrementally update the net-cut and amortize the amount of time used to a constant number of operations per net over the entire bin. We do the same with the centers of the cells in the bin to incrementally update the cell areas on either

side of the cut-line as well as the induced partitioning. While processing each cut-line, we save the cut-line with smallest cut that is legal given partitioning tolerances. An example of finding the cut-line for a partitioning bin is shown in Figure 6.3.

Once a partitioning has been chosen, we accept or reject it based on how much it can be improved by **a single pass of a Fiduccia-Mattheyses partitioner with early termination** (which takes only several seconds even on the largest ISPD'05 circuit).[1] The intuition is that if the constructed partitioning is not worthy of reuse, a single Fiduccia-Mattheyses pass could improve its cut non-trivially. If the Fiduccia-Mattheyses pass improves the cut beyond a certain threshold, we discard the solution and bisect the entire bin from scratch. If this test passes, we check legality: if a child bin is overfull, we discard the cut-line and bisect from scratch.

6.3.3 Scalability

Pseudocode for the cut-line location process used by ECO-system is shown in Figure 6.4. The runtime of the algorithm is linear in the number of pins incident to the bin, cells incident contained in the bin, and possible cut-lines for the bin. Since a single Fiduccia-Mattheyses pass takes also takes linear time [59], the asymptotic complexity of our algorithm is linear. If we let P represent the number of pins incident to the bin, C represent the number of cells in the bin and L represent the number of potential cut-lines in the bin, the cut-line selection process runs in $O(P + C + L)$ time. In the vast majority of cases, $P > C$ and $P > L$, so the runtime estimate simplifies to $O(P)$.

The number of bins may double at each hierarchy layer, until bins are small enough

[1] We do not assume that the initial placement was produced by a min-cut algorithm.

ALGORITHM 6.2: Linear-time vertical cut-line selection

```
    ▷ Input: placement bin, balance constraints
    ▷ Output: x-coord of best cut-line, BESTX
 1  numCutlines ← 1 + ⌊(rightBinEdgeX − leftBinEdgeX)/ cellSpacing⌋
 2  Create three arrays of size numCutlines: LEFT, RIGHT, AREA
 3  Set all the elements of LEFT, RIGHT and AREA to 0
 4  foreach(net incident to bin)
 5      do Calculate x-coord of left- and right-most pins on net: leftPinX and rightPinX
 6         leftCutlineIndex ← MAX(0, ⌈(leftPinX −idleftBinEdgeX)/ cellSpacing⌉)
 7         rightCutlineIndex ← MAX(0, ⌈(rightPinX −idleftBinEdgeX)/ cellSpacing⌉)
 8         if (leftCutlineIndex < numCutlines)
 9           then LEFT[leftCutlineIndex]++
10         if (rightCutlineIndex < numCutlines)
11           then RIGHT[rightCutlineIndex]++
12         foreach(cell on net)
13             do centerX ← x-coord of the center of cell
14                cutlineIndex ← MAX(0, ⌈(centerX − leftBinEdgeX)/ cellSpacing⌉)
15                if (cutlineIndex < numCutlines)
16                  then AREA[cutlineIndex] + = GETCELLAREA(cell)
17         X ← leftBinEdge, CURCUT ← 0, BESTCUT ← ∞,
18         BESTX ← ∞, LEFTPARTAREA ← 0
19         for (I ← 0; I < numCutLines; I++, X + = cellSpacing)
20             do CURCUT + = LEFT[I]
21                CURCUT − = RIGHT[I]
22                LEFTPARTAREA + = AREA[I]
23                if (CURCUT < BESTCUT and LEFTPARTAREA satisfies constraints)
24                  then BESTCUT ← CURCUT
25                       BESTX ← X
26  return BESTX
```

Figure 6.4: Algorithm for finding the best vertical cut-line from a placement bin. Finding the best horizontal cut-line is largely the same process. Note that the runtime of the algorithm is linear in the number of pins incident to the bin, cells contained in the bin, and possible cut-lines for the bin.

for end-case placement. End-case placement is generally a constant amount of runtime for each bin, so it does not affect asymptotic calculations. Assume that ECO-system is able to reuse all of the original placement. Since ECO-system performs bisection, it will have $O(\log C)$ layers of bisection before end-case placement. At layer i, there will be $O(2^i)$ bins, each taking $O\left(\frac{P}{2^i}\right)$ time. This gives a total time per layer of $O(P)$. Combining all layers gives $O(P \log C)$. Empirically, the runtime of the cut-line selection procedure (which includes a single pass of a Fiduccia-Mattheyses partitioner) is much smaller than

partitioning from scratch. On large benchmarks, cut-line selection requires 5% of ECO-system runtime time whereas min-cut partitioning generally requires 50% or more of ECO-system runtime.

6.3.4 Handling Macros and Obstacles

With the addition of macros, the flow of top-down placement becomes more complex. We adopt the technique of "floorplacement" which proceeds as traditional placement until a bin satisfies criteria for block-packing [107, 115]. If the criteria suggest that the bin should be packed rather than partitioned, a fixed-outline floorplanning instance is induced from the bin where macros are treated as hard blocks and standard cells are clustered into soft blocks. The floorplanning instance is given to a Simulated Annealing-based floorplanner to be solved. If macros are placed legally and without overlap, they are considered fixed. Otherwise, the placement bin is merged with its sibling bin in the top-down hierarchy and the merged bin is floorplanned. Merging and re-floorplanning continues until the solution is legal.

We add a new floorplanning criterion for our legalization technique. If no macros in a placement bin overlap each other, we generate a placement solution for the macros of the bin to be exactly their placements in the initial solution. If some of the macros overlap each other, we let other criteria for floorplanning decide. If block-packing is invoked, we must discard the placement of all cells and macros in the bin and proceed as described in [115].

During the cut-line selection process, some cut-line locations are considered invalid — namely those that are too close to obstacle boundaries but do not cross the obstacles.

This is done to prevent long and narrow slivers of space between cut-lines and obstacle boundaries. Ties for cut-lines are broken based on the number of macros they intersect. This reduces overfullness in child bins allowing deeper partitioning, reducing runtime.

6.3.5 Controlling Overlaps, Whitespace and Congestion

We introduce techniques and user controls for ECO-system and show how they can be used for reallocation of whitespace and congestion improvement in the original placement.

Relaxing overfullness constraints. One of the primary objectives of ECO-system is to reuse as much relevant placement information as possible from a given placement. As described above, it is possible to find a cut-line which has a good cut but is not legal due to space constraints. In these cases, ECO-system must discard these good solutions and partition from scratch. In order to make better use of the given placement, we propose the following addition to ECO-system. We allow ECO-system to shift the cut-line to legalize the derived partition with respect to area. Cut-line shifting is a technique commonly used in the top-down min-cut placement for allocation of whitespace [5, 92, 93, 116, 120]. The cut-line is shifted as little as possible to make the derived partitioning legal with respect to area. If it is impossible to find an area-legal cut-line, the derived partitioning must be discarded and ECO-system proceeds normally.

If cut-line shifting is successful in correcting the illegality, the original placement must be modified for purposes of consistency. To do so, cells are scaled proportionately within the placement bin based on their original positions, the position of the originally chosen cut-line and the position of the shifted cut-line in a manner similar to that in the WSA technique [92, 93]. As the centers of cells are used to determine in what partitions cells

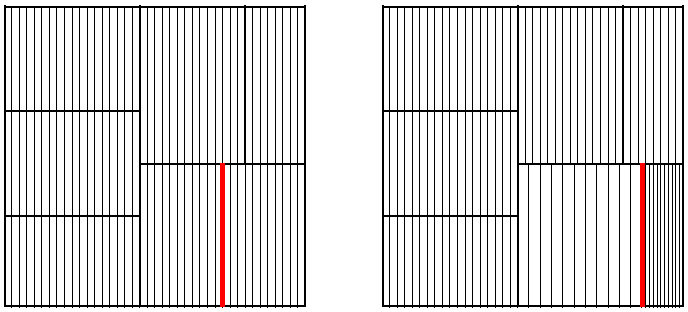


Figure 6.5: Shifting a cut-line chosen during ECO cut-line selection. Unlike the WSA technique [92, 93], cut-line shifting during ECO is not done on geometric cut-lines but instead on those cut-lines which are chosen during fast cut-line selection. The image on the left shows a placement that has been divided into bins during the course of ECO-system. In the image on the right, the chosen cut-line of the bottom-right bin is shifted to the right. The density of vertical lines represents the initial placement and its scaling around the moving cut-line (shown in red).

belong during fast cut-line selection, we shift cell locations based on center locations as well to ensure that cut-line shifting will not change derived partitions. We seek to shift cell locations and maintain the following property: the relative position between cells before and after shifting is maintained. Also, if a cell were in the middle of a partition before shifting, it should remain in the middle of a partition after shifting. Let x_L and x_R represent the x-coordinates of the left and right sides of the placement bin, x_{orig}^{cut} and x_{new}^{cut} the x-coordinates of the original and new cuts, and, lastly, x_{orig}^{cell} and x_{new}^{cell} the x-coordinates of the center of a particular cell before and after shifting. We wish to maintain the following ratios (for vertical partitioning):

$$\begin{aligned}
\frac{x_{orig}^{cell} - x_L}{x_{orig}^{cut} - x_L} &= \frac{x_{new}^{cell} - x_L}{x_{new}^{cut} - x_L},\ x_{orig}^{cell} \leq x_{orig}^{cut} \\
\frac{x_R - x_{orig}^{cell}}{x_R - x_{orig}^{cut}} &= \frac{x_R - x_{new}^{cell}}{x_R - x_{new}^{cut}},\ x_{orig}^{cell} > x_{orig}^{cut}
\end{aligned}$$

Solving for x_{new}^{cell} :

$$x_{new}^{cell} = \begin{cases} x_L + \left(x_{orig}^{cell} - x_L\right) \frac{x_{new}^{cut} - x_L}{x_{orig}^{cut} - x_L}, & x_{orig}^{cell} \leq x_{orig}^{cut} \\ x_R - \left(x_R - x_{orig}^{cell}\right) \frac{x_R - x_{new}^{cut}}{x_R - x_{orig}^{cut}}, & x_{orig}^{cell} > x_{orig}^{cut} \end{cases}$$

Y-coordinates of cells shifted during horizontal partitioning are calculated analogously.

Figure 6.5 illustrates the scaling involved when a cut-line is shifted. In the figure, the cut-line of the bottom-right bin is shifted to the right. All objects to the left and right of the cut-line are scaled appropriately. Objects that were to the left of the original cut-line remain to the left and are spread out and objects on the right are packed closer together.

Shifting proportionately in this way maintains the relative ordering of all the cells within the current placement bin. Also the partitioning induced by the cut-line remains unchanged so ECO-system can proceed as normal. Shifting the cut-line in this manner can allow deeper ECO partitioning which can reduce both runtime and cell displacement.

Satisfying density constraints. A common method for increasing the routability of a design is to inject whitespace into regions that are congested [5, 93]. One can also require a minimum amount of whitespace (equivalent to a maximum cell density) in local regions of the design to achieve a similar effect [120]. As one of ECO-system's legality checks is essentially a density constraint (checking to see if a child bin has more cell area assigned to it than it can physically fit), this legality check is easy to generalize. The new criterion for switching from using the initial placement and partitioning from scratch is based on a child bin having less than a threshold percent of relative whitespace, which is controlled by the user.

The cut-line shifting feature of ECO-system can also be used to satisfy density constraints. As ECO-system proceeds, cut-lines can be shifted as described above to imple-

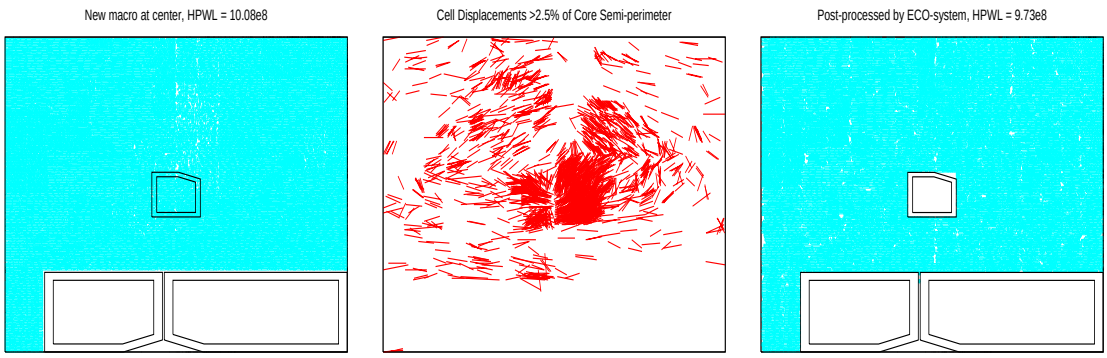


Figure 6.6: Legalizing the placement of a new fixed obstacle at the center of the ICCAD'04-Faraday design DSP1 [1]. The picture in the middle shows the movement of standard cells to make room for the obstacle. Many standard cells must move in order to accommodate the obstacle, but ECO-system moves these cells on average only a short distance (1.27% of the core half-perimeter) and is able to improve total HPWL.

ment a variety of whitespace allocation schemes [92, 93, 116, 120]. Specifically, ECO-system can implement the hierarchical whitespace injection of WSA [92, 93]. WSA chooses cut-lines based only on the geometry of a placement bin and shifts these cut-lines from the top down. ECO-system chooses cut-lines that are more natural to the original placement, shifts cut-lines top-down, and also supports fixed objects and movable macros. Figure 6.1 shows the power of the cut-line shifting technique in redistributing whitespace for routability after making a change to a placement that causes significant overlap.

6.4 Using ECO-system in High-level and Physical Synthesis

We extend the proposed framework to offer users efficient access to the features of incremental placement described in Sections 6.2 and 6.3 as well as provide greater user control and flexibility.

6.4.1 Additional User Controls

We present further controls over ECO-system to vary how much it is allowed to modify a given placement as well as what regions of a placement are allowed to be changed, which can both be beneficial to a designer. We also illustrate how ECO-system can be used to re-optimize placements based on changes to net weights. This control can be extremely useful when critical nets in a design change, for example.

Tunable aggressiveness. ECO-system accepts or rejects derived partitioning solutions based on how much a single pass of a Fiduccia-Mattheyses partitioner can improve them. If the partitioner improves the net cut by more than a threshold percentage, the partitioning solution is rejected. This threshold can be adjusted by the user so as to prevent ECO-system from performing large changes. If a designer wants ECO-system to change the placement as little as possible, the improvement threshold can be given as 100%. Tunable aggressiveness also allows one to adjust the strength of ECO-system legalization to better correlate with the magnitude of design modifications [76].

User-defined locality. ECO-system operates automatically on the given placement and quickly focuses on sections of overlap. It may be the case that a designer has performed optimization on only a small portion of the design. Having our algorithm run over the entire design to find this small area is potentially wasteful. Thus we allow the user or a physical synthesis tool to specify one or more regions of the placement area to apply legalization. Combined with whitespace control techniques described above, this allows a designer to re-tune whitespace allocation to reduce congestion in localized regions.

While this control can be useful to designers to ensure that certain regions of a design

remain untouched, it is not a replacement for the automatic techniques of ECO-system. Changes made to one region of a design can affect the quality of the placement in a separate area of the design. Patch-based replacement of a design does not handle this situation well because the patches must be supplied but may not be well-defined. Also, the processing of given patches in a particular order can make the legalization within the first patch inconsistent with that in subsequent patches. However, ECO-system can automatically narrow down the regions that require extra work, partition them, and simultaneously perform top-down legalization in all regions to ensure consistency. Cut-line shifting in ECO-system is truly hierarchical and allows ECO-system to subsume other hierarchical techniques such as WSA [92, 93] while also supporting fixed objects and movable macros.

Changing net weights. Having a legal placement facilitates more precise static timing analysis and finding timing-critical nets. To improve timing, weights can be increased for nets with worst slack, and decreased for non-critical nets. As ECO-system checks if the cut of an induced partitioning solution can be improved significantly, net weights are naturally integrated into this test. With weighted cut, ECO-system recognizes instances when the initial placement can be improved.

6.4.2 Placing New Cells and Macros

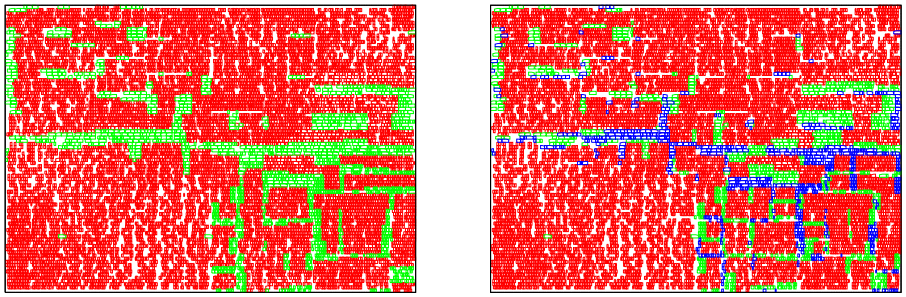
The addition of macros, IP blocks and embedded memories to an already placed netlist can introduce significant overlap as can be seen in Figure 6.6. Large modules may need to be fixed due to alignment constraints and will appear as obstacles. Buffer insertion is also a concern as numerous buffers may need to be inserted. There are typically few legal locations for buffer insertion, and, compounding the problem, buffers must be placed

precisely to be effective.

Our current technique can accommodate newly added modules for which tentative initial placements are given. All a designer would need to do is place new modules roughly where they should go in the core, and ECO-system will find legal positions for them automatically. If new module locations are not known, they can be found with simple analytical techniques. Specifically, if an unplaced module is connected to several placed modules, an initial location for the module could be the average location of its neighbors. This does not work well, however, when a cluster of new logic is added to a design, especially in the presence of macros and obstacles. For this reason, we develop a technique to place unplaced modules within ECO-system.

To handle new modules separately, one must be able to detect them easily in a design. Some input formats allow the user to specify modules which are new with the keyword UNPLACED. For other input formats without such a keyword, ECO-system checks for modules that are placed outside of the core and marks them as being unplaced. ECO-system also tests to see if several modules are placed at exactly the same location which could indicate a cluster of new logic. Modules placed in exactly the same location, such as a default location like (0,0), are also treated as unplaced.

In each bin, if a cut-line and partitioning are derived, unplaced modules are partitioned with a separate partitioning call to assign them to child bins. If the derived partitioning is not accepted, unplaced modules are combined with the old modules, and placement continues from scratch. In this way, unplaced modules will migrate to good legal locations automatically. As the locations for unplaced modules are chosen based on current loca-



(a) (b)

Figure 6.7: Using ECO-system to perform body bias clustering. ECO-system refines an initial placement and moves cells with the same bias into contiguous regions to reduce the area overhead of adaptive body biasing while preserving interconnect length. Cells are grouped into (a) 2 and (b) 3 bias clusters based on their power characteristics in an initial placement. Cells with the same bias share the same color.

tions of all the modules in the design, the final locations of unplaced modules will likely be better than ones that were chosen based on the initial placement.

If new modules are introduced into a design and a user defines a region of the placement to work in, there is some ambiguity in what ECO-system should do with unplaced modules. All unplaced modules could be placed inside the user-specified region, or ECO-system could determine which of the unplaced modules would best be placed in the region. Determining which of the unplaced modules belong in a user-specified rectangular region requires at most four calls to a partitioner (since the region can be carved out with four geometric cut-lines), so this will still be efficient. To avoid uncertainty, the user is allowed to specify which behavior is desired.

6.4.3 An Application to Body Bias Clustering

Adaptive body biasing (ABB) is a technique that allows one to tune individual manufactured dies to optimally meet delay and power constraints. ABB compensates for within-

die parameter variations by allowing one to apply optimal body bias voltages to groups of cells on a die [133]. One can forward bias device bodies to decrease Vt leading to higher drive currents and smaller delay or reverse bias device bodies leading to higher Vt and lower leakage power.

One especially important matter to consider when applying ABB is the area overhead associated with adjacent cells of different bias. Adjacent cells with different bias must observe strict spacing rules. Thus if several cells with different biases are placed next to each other, they will need to be spaced out incurring a significant area overhead. Recently, a technique was proposed to group standard cells into a small number of clusters each having the same bias [88]. These clusters are determined by examining the power characteristics in a given initial placement. By moving cells which belong to the same cluster toward each other, area overhead can be saved while still achieving power and delay constraints.

ECO-system has been adapted to reduce the area overhead for such body bias clustered placements in the following way. ECO-system proceeds until a user-controllable size threshold on the number of cells in a placement bin is reached. When the number of cells in the bin is less than the threshold, ECO-system finds which clusters are represented in the bin. If all of the cells in the bin belong to the same cluster, ECO-system proceeds as described in Section 6.3. If this is not the case, ECO-system evaluates several partitionments for the given bin and chooses the one with best balanced-cut. The partitionments evaluated are the ones where each unique cluster is partitioned away from the remainder of the cells. For example, if we are partitioning vertically, each cluster can go either on the left or right of the cut-line. In the special case of $n = 2$, only 2 partitionments need

to be evaluated; when $n > 2$ ECO-system evaluates $2n$ partitionments. After the best partitionment is chosen, child bins are created and marked to be placed from scratch.

Figure 6.7 depicts two placements that have been modified by ECO-system to spatially cluster cells with the same bias. Cells with the same bias have the same color. In these placements, only 4% of cells are moved and HPWL is increased by 2.3% and 3.1%. The area overhead ranges from 5.2-7.8% which is outweighed by the improvements to power and delay afforded by the biasing [88].

6.5 Empirical Results

We implemented ECO-system in C++ and ran it on 3.2GHz Pentium Xeon machines. In this section we present results dealing with the legalization of benchmarks altered due to cell resizing, the effect of ECO-system on the timing of resized benchmarks, and using ECO-system to legalize various analytically-generated global placements.

6.5.1 Legalization of Resized Netlists

For testing we use three suites of benchmarks. The first suite of benchmarks are the IC-CAD 2004 IBM-MSwPins benchmarks: mixed-size netlists with non-trivial macro sizes, aspect ratios and pin offsets [1]. We placed all of the benchmarks with Capo 10 [120] and chose the best of 2 runs. Next we randomly resized the standard cells of the benchmark to simulate cell sizing such that the total area of cells would remain relatively constant. Each standard cell of the design was randomly increased or decreased in size, but no cell was decreased below the minimum cell size or increased beyond the largest cell size.

The change in cell area and amount of overlap introduced by the resizing is shown in

Table 6.2: Overlap legalization on the IBM-MSwPins [1] and ISPD'05 Contest benchmarks [104]. "Area Ratio" represents the change in total cell area after resizing. Overlap is measured as % of the total movable cell and macro area. Full data for the ISPD'05 benchmarks can be found in [117]. ECO-system requires significantly more runtime than the Capo 10 legalizer [115], and approximately 16% of the original placement time. ECO-system increases HPWL by 0.61% on average while the Capo 10 legalizer increases HPWL by 3.93% on the IBM-MSwPins benchmarks. On the ISPD'05 Contest benchmarks ECO-system *decreases* HPWL by 1.00% on average while the Capo 10 legalizer increases HPWL by 4.28%.

IBM-MSwPins	Area	Orig.	Orig.	Overlap	Capo 10 Legalizer [115]			ECO-system			ECO-system /w shifting		
Benchmarks	Ratio	Time (s)	HPWL		Time (s)	HPWL	Ratio	Time (s)	HPWL	Ratio	Time (s)	HPWL	Ratio
ibm01	0.9982	248	2.48	7.35%	1.27	2.57	1.0371	44.4	2.48	0.9995	45.2	**2.47**	0.9957
ibm02	1.0008	463	5.12	5.56%	2.15	5.28	1.0328	77.3	5.13	1.0024	81.0	**5.11**	0.9980
ibm03	1.0011	661	7.58	5.83%	15.9	7.99	1.0543	128	7.54	0.9951	127	**7.53**	0.9934
ibm04	0.9990	728	8.61	8.13%	11.3	9.03	1.0482	149	8.67	1.0070	147	**8.66**	1.0055
ibm05	1.0017	593	10.14	13.54%	0.13	10.25	1.0114	141	10.32	1.0177	149	**10.28**	1.0139
ibm06	1.0018	846	6.78	7.36%	10.5	7.10	1.0469	152	6.82	1.0054	155	**6.79**	1.0019
ibm07	0.9997	1213	11.63	9.61%	16.4	12.16	1.0455	201	11.72	1.0081	210	**11.69**	1.0052
ibm08	1.0029	1492	13.42	8.50%	7.36	13.73	1.0232	211	13.54	1.0090	223	**13.49**	1.0054
ibm09	1.0025	1492	14.96	8.14%	14.8	16.06	1.0732	288	14.89	0.9954	296	**14.82**	0.9907
ibm10	0.9997	2476	31.79	4.53%	119	32.62	1.0260	387	31.54	0.9922	390	**31.48**	0.9903
ibm11	0.9993	2067	21.43	8.48%	26.3	22.56	1.0529	384	21.63	1.0092	411	**21.44**	1.0005
ibm12	0.9996	2903	38.52	5.91%	50.6	39.20	1.0175	379	37.95	0.9851	393	**37.82**	0.9819
ibm13	1.0014	2667	27.30	7.94%	55.3	28.61	1.0478	586	27.57	1.0101	587	**27.31**	1.0004
ibm14	1.0002	4954	40.00	13.49%	38.3	41.67	1.0417	734	40.70	1.0174	744	**40.58**	1.0144
ibm15	1.0016	6241	53.72	10.85%	63.1	56.48	1.0514	1127	54.68	1.0178	996	**54.68**	1.0178
ibm16	0.9997	7232	61.12	9.19%	36.2	62.74	1.0264	890	61.42	1.0050	907	**61.20**	1.0014
ibm17	0.9987	7558	70.52	14.09%	36.0	73.09	1.0365	983	71.65	1.0160	1009	**71.45**	1.0132
ibm18	1.0017	6897	46.46	15.91%	13.7	48.11	1.0354	1006	47.30	1.0182	1032	**47.13**	1.0145
Average	1.0005	1.0000			0.0102		1.0393	0.1551		1.0061	0.1558		1.0024

ISPD'05	Area	Orig.	Orig.	Overlap	Capo 10 Legalizer [115]			ECO-system			ECO-system /w shifting		
Benchmarks	Ratio	Time (s)	HPWL		Time (s)	HPWL	Ratio	Time(s)	HPWL	Ratio	Time (s)	HPWL	Ratio
adaptec1	1.0004	9403	83.87	18.17%	1020	88.81	1.0589	1686	83.96	1.0011	1454	**83.58**	0.9965
adaptec2	1.0012	9978	87.31	16.83%	1246	91.48	1.0477	2138	88.82	1.0173	1608	**88.31**	1.0114
adaptec3	1.0004	26937	231.17	17.37%	3090	240.44	1.0401	4283	**224.75**	0.9722	3762	225.23	0.9743
adaptec4	1.0005	29266	187.65	16.81%	1775	194.89	1.0386	3759	**189.73**	1.0111	3474	189.86	1.0119
bigblue1	1.0005	10752	101.96	15.62%	1.6	104.77	1.0276	1535	101.41	0.9946	1501	**101.30**	0.9936
bigblue2	0.9994	27902	159.08	16.15%	1238	164.21	1.0322	4456	158.07	0.9937	4271	**157.91**	0.9926
bigblue3	0.9999	69498	414.29	15.69%	4169	445.95	1.0764	8402	**391.05**	0.9439	7547	401.10	0.9682
bigblue4	1.0006	118741	884.39	15.58%	953	903.81	1.0220	11072	**873.48**	0.9877	10795	874.10	0.9884
Average	1.0004	1.0000			0.0415		1.0428	0.1234		0.9899	0.1138		0.9920

Table 6.2. The resized benchmarks should have legal placements with HPWL near that of the original benchmarks since total cell area does not change appreciably. Discussions with colleagues in the industry point out that cell resizing is affected by a variety of factors, which are not as random as in our experiments. The IBM-MSwPins benchmarks do not contain enough information to perform more intelligent resizing, so this experiment is used primarily to evaluate ECO-system in the presence of many movable macros.

We compare ECO-system to the legalizer of Capo 10, and the results are summarized in Table 6.2. The Capo legalizer runs quickly and produces legal placements, but it increases HPWL by 3.93% on average. ECO-system takes less than 16% of the original placement time, and only increases HPWL by 0.61% on average. By adding cut-line shifting to ECO-system runtime is largely unaffected but the HPWL increase is further reduced to 0.24%. We have also varied the amount of overlap introduced into these benchmarks by reducing the number of cells affected by our sizing. We find that HPWL is relatively unaffected (HPWL generally changes by less than 0.5%) by increasing amounts of overlap for these designs.

The second set of benchmarks are from the ISPD 2005 Placement Contest [104]. They are a standard cell benchmark suite with non-trivial fixed obstacles throughout the placement area [104]. We placed all of the benchmarks with APlace 2.04 [82] (the winning placer of the contest) and randomly resized the standard cells of the benchmark in the same way as the IBM-MSwPins benchmarks as the ISPD 2005 benchmarks do not contain necessary information for more intelligent resizing. As a result, the focus of this experiment is to see how ECO-system performs on very large-scale placement instances in the presence fixed obstacles.

The change in cell area and amount of overlap introduced by the resizing is shown in Table 6.2. A comparison of ECO-system to the legalizer of Capo 10 is summarized in Table 6.2. Full data for the ISPD'05 benchmarks can be found in [117]. The Capo legalizer runs 40% faster than ECO-system, but increases HPWL by 4.28% on average. ECO-system takes 14% of the original placement time, and *decreases* HPWL by 1.00%.

Table 6.3: Overlap legalization on the IWLS 2005 Benchmarks [72]. "Area Ratio" represents the change in total cell area after resizing. Overlap is measured as % of the total movable cell area. ECO-system decreases HPWL by 1.81% on average while the Capo 10 legalizer increases HPWL by 1.85%.

IWLS Benchmarks	Area Ratio	Orig. Time (s)	Orig. HPWL	Overlap	Capo 10 Legalizer [115]			ECO-system		
					Time (s)	HPWL	Ratio	Time (s)	HPWL	Ratio
aes_core	1.0278	519	23.70	14.30%	0.2	23.91	1.0089	64.4	**22.94**	0.9679
ethernet	1.1122	3666	105.71	13.34%	0.5	108.73	1.0286	284	**104.78**	0.9912
mem_ctrl	1.0508	404	16.29	13.24%	0.1	16.63	1.0209	32.6	**15.95**	0.9791
pci_bridge32	0.9724	550	19.61	11.27%	0.2	20.09	1.0245	55.8	**19.21**	0.9796
usb_funct	1.0901	346	15.93	13.82%	0.1	16.34	1.0257	39.3	**15.72**	0.9868
vga_lcd	0.9841	15686	370.79	9.06%	1.1	371.76	1.0026	819	**365.87**	0.9867
Average	1.0383	1.0000			0.0001		1.0185	0.0612		0.9819

Figure 6.8 depicts the benchmark adaptec3 before cell resizing and after legalization with ECO-system. ECO-system's placement is similar to the original APlace 2.04 placement and does not move the majority of cells far from their original locations. The average displacement per cell is 0.28% of the half-perimeter of the design which is an order of magnitude less than WSA's displacements [92, 93] and those reported in [5]. Only 1.98% of the cells have non-trivial displacements.

The third set of benchmarks on which we perform experiments with resizing is the IWLS 2005 suite of benchmarks [72]. These benchmarks contain information such as the signal directions of pins, so we were able to resize cells in a more realistic way based on wire load. The benchmarks were first placed using Capo 10 in ROOSTER mode [116] for routability. Next, for each cell we calculated the Steiner length of wires the cell drives. According to the theory of logic effort, longer wires should be driven by larger cells [127], so we increased the sizes of cells whose driven lengths were longer than the median driven length and decreased the size of cells whose driven lengths were shorter than the median. The amount of overlap introduced by this resizing method is shown in Table 6.3. We compare ECO-system to the Capo 10 legalizer and find again that the Capo 10 legalizer

Table 6.4: Overlap legalization of APlace 2.04's [82] global placements of the ISPD'05 Contest benchmarks [104]. Overlap is measured as % of the total movable cell area. ECO-system produces legal solutions with nearly the same or better HPWL than APlace 2.04's legalizer. APlace's legalizer increases HPWL by 4.91% while ECO-system increases HPWL by 3.68% and only 2.35% when using shifting. ECO-system with shifting is faster on 7 of the 8 benchmarks and four times faster than APlace's legalizer overall.

Benchmark	Orig.	Illegal	Overlap	APlace 2.04 Legalizer [82]			ECO-system			ECO-system /w shifting		
	Time (s)	HPWL		Time (s)	HPWL	Ratio	Time (s)	HPWL	Ratio	Time (s)	HPWL	Ratio
adaptec1	7569	81.05	34.74%	1346	83.87	1.0348	1656	85.17	1.0508	1386	**82.23**	1.0146
adaptec2	6062	94.22	47.25%	2543	101.64	1.0788	2037	101.10	1.0730	1684	**97.85**	1.0385
adaptec3	15849	211.13	47.12%	11495	231.17	1.0949	4245	227.25	1.0763	3672	**222.24**	1.0526
adaptec4	15404	197.24	36.78%	15271	206.23	1.0456	3805	202.26	1.0255	3505	**200.80**	1.0180
bigblue1	8265	100.51	28.53%	2486	**101.96**	1.0144	1607	104.22	1.0369	1262	102.50	1.0198
bigblue2	13650	154.51	30.15%	14252	159.08	1.0296	3882	156.35	1.0119	3840	**155.83**	1.0086
bigblue3	30624	385.40	41.06%	38873	414.29	1.0750	12546	**386.99**	1.0041	10080	395.11	1.0252
bigblue4	61932	865.03	32.01%	56809	884.39	1.0224	11552	880.58	1.0180	10451	**874.90**	1.0114
Average	1.0000			0.8978		1.0491	0.2594		1.0368	0.2252		1.0235

is extremely fast, but increases HPWL significantly (1.85%) while ECO-system is able to *reduce* HPWL by 1.81% on average. For this experiment we did not use ECO-system's cut-line shifting feature in order to preserve Capo's routability-driven whitespace allocation.

6.5.2 ECO-system's Impact on Timing

One of the most important goals of an incremental placer is to preserve the timing characteristics of a design after timing optimizations have been performed on the design. Recall that cell sizing and buffer insertion decisions are based on circuit timing. If an incremental placer moves cells too drastically, popular timing optimizations can be less effective and eventually degrade timing rather than improve it. Therefore, we evaluate the impact of ECO-system on circuit timing. For these experiments we resized the 20 of the OpenCores designs that were part of the IWLS 2005 benchmark suite [72] in a realistic manner (as described above) and evaluate timing characteristics of the resized netlists before and after legalization by ECO-system.

Circuit timing was evaluated using a Static Timing Analysis engine which uses the D2M net delay model (more accurate then Elmore) [8] for each net based on Steiner trees produced by the FLUTE package [44]. The worst change in circuit delay for these designs was an increase of 8.07%. The average change was 1.00% while the best was a decrease of 7.37% of maximum delay. Thus ECO-system is effective in preserving the timing of a netlist by minimally impacting maximum delay during legalization and in some cases can further improve it. In this experiment ECO-system is completely independent of the timing analyzer used and therefore our results are likely to hold for other STA engines.

6.5.3 Legalizing Analytical Global Placements

Analytical placements generally contain a significant amount of overlap after global placement, especially so on the ISPD'05 Contest benchmarks given their numerous fixed obstacles in the core region. Therefore, we compare ECO-system to the APlace 2.04 legalizer on APlace 2.04 global placements on the ISPD'05 Contest benchmarks. Table 6.4 shows that APlace 2.04 global placements have overlap of approximately 30% or more. APlace 2.04's legalizer generally increases HPWL by 4.91% while ECO-system increases HPWL only 3.68% on average. ECO-system is also three times faster than APlace's legalizer. Adding cut-line shifting improves ECO-system's results, increasing HPWL by only 2.35% while running four times faster than APlace's legalizer.

To illustrate the effectiveness of ECO-system in redistributing whitespace to improve routability, we placed the IBMv2 benchmark suite [143] with the analytical placer mPL6 [27]. mPL6 global placements were refined by ECO-system and then routed using Cadence WarpRoute. In Table 6.5 we compare the placements refined by ECO-system to

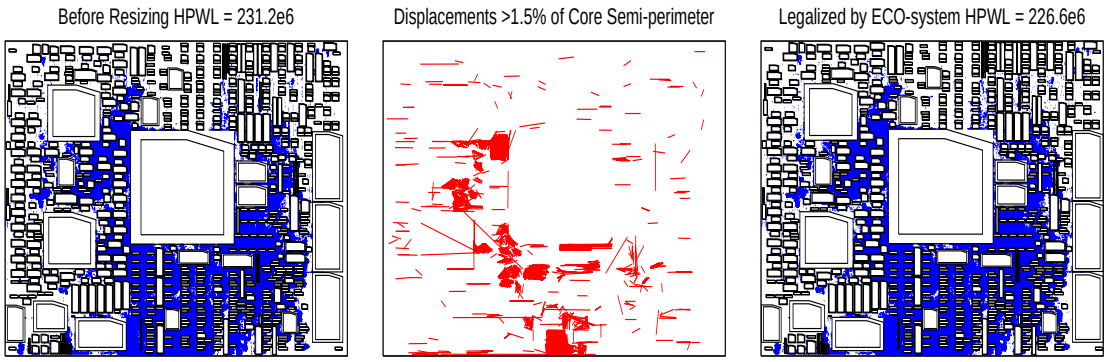


Figure 6.8: When applied to resized netlist, ECO-system produces a placement (right) similar to the original placement (left). Fixed objects are outlined in double black lines. The largest cell displacements are shown in red (center). Only displacements larger than 1.5% of the half-perimeter of the design are shown. Average displacement is 0.28% of the design half-perimeter. The majority of the large displacements form around the corners of the large, fixed obstacles. Many of these large displacements appear to be clustered, indicating small groups of modules transported to another region of the core or spread to accommodate area increases.

those produced by the detailed placer of mPL6 (XDP [52]) in terms of routed wirelength (Rt WL), via counts, violations, and routing time (Rt Time). ECO-system improves mPL6 global placements to the point where the router completes in all cases, reducing routed wirelength by 1.1%, via counts by 7.8% and routing time by more than half on average.

To test the ECO-system's routability improvements in the presence of fixed obstacles, we use the ICCAD'04-Faraday benchmarks [1]. The Faraday benchmarks are a suite of mixed-size benchmarks with routing information based on netlists released by the Faraday Corporation [1]. For our experiments, we fix macros to their original locations as determined by Silicon Ensemble Ultra v5.4.126 (details on the construction of the benchmarks can be found in Appendix A of [1]). We run mPL6 on the four benchmarks with macros and produce global and detailed placements of each. As in the previous experiment, we compare mPL6 detailed placements to mPL6 global placements refined by ECO-system.

Table 6.5: Improving the routability of analytical placements using ECO-system. We compare the routability of mPL6 [27] global placements when using mPL6's detailed placer (XDP [52]) vs. ECO-system with cut-line shifting for detailed placement on the IBMv2 benchmark suite [143]. Best legal routed wirelength (Rt WL) and via counts are highlighted in bold. ECO-system produces routable placements in all cases, reduces routed wirelength by 1.1% and via counts by 7.8%, and cuts routing runtime by more than half.

Benchmark	XDP [52]				ECO-system			
	Rt WL	Vias	Viols.	Rt Time (m)	Rt WL	Vias	Viols.	Rt Time (m)
ibm01e	723961	150166	806	1052	**745660**	**125177**	0	22
ibm01h	735409	156414	348	654	**701959**	**122995**	0	70
ibm02e	1937102	261495	0	27	**1822638**	**247396**	0	13
ibm02h	2004969	324609	108	133	**1933310**	**255647**	0	18
ibm07e	3817994	497500	0	54	**3555210**	**468105**	0	22
ibm07h	3814735	569897	49	91	**3658097**	**479911**	0	25
ibm08e	3999658	587627	0	31	**3970074**	**561636**	0	24
ibm08h	3948739	591744	0	35	**3914580**	**574135**	0	28
ibm09e	**2891305**	483046	0	17	2956856	**472863**	0	17
ibm09h	**2935006**	490682	0	19	2965823	**480363**	0	18
ibm10e	**5753519**	773695	0	36	5888185	**750270**	0	30
ibm10h	**5742241**	778756	0	35	5762900	**759962**	0	31
ibm11e	**4399838**	637627	0	26	4438438	**615691**	0	23
ibm11h	4670094	645872	0	31	**4634023**	**630791**	0	25
ibm12e	**8640070**	972714	0	66	8697654	**908164**	0	42
ibm12h	**8695922**	977498	0	69	8726583	**926119**	0	53
Ratio	1.000	1.000		1.000	0.989	0.922		0.446

Results for this experiment are shown in Table 6.6. ECO-system placements are mostly routable with a few violations, but the mPL6 placements are completely unroutable. We were unable to run the WSA technique on the mPL6 placements as WSA does not support fixed obstacles, which we have confirmed with the authors of WSA.

6.6 Conclusions

Below we summarize our work, outline several additional applications and articulate our contributions to shared infrastructure for research in placement.

Table 6.6: Improving the routability of analytical placements in the presence of fixed obstacles in the ISPD'04-Faraday benchmark suite [1]. We postprocess mPL6 [27] global placements using mPL6's detailed placer and, separately, our ECO-system (with cut-line shifting). The mPL6 detailed placer XDP [52] produces largely unroutable placements.

Bench-mark	XDP [52]				ECO-system			
	Rt WL	Vias	Viols.	Rt Time (m)	Rt WL	Vias	Viols.	Rt Time (m)
dsp1	1041556	233408	112883	12	1162096	202700	0	6
dsp2	-	-	-	>24 hrs.	1117349	201598	0	6
risc1	2042695	342856	373088	71	2066426	344258	10	10
risc2	-	-	-	>24 hrs.	1906434	337809	11	11

6.6.1 Summary of Our Work

We have presented ECO-system — an algorithmic framework designed to interface a wide variety of circuit optimizations with their physical environment. This framework offers, for the first time in the literature, a strong and robust legalizer that can handle a broad range of modern placement instances with movable macros, fixed obstacles, etc. ECO-system automatically focuses on regions of the layout and sections of the netlist that require changes, and performs optimization of adequate strength in each case. ECO-system can be combined with an external global placer invoked when particularly large changes are required. It can also be used in incremental resynthesis, in high-level and physical synthesis optimizations, and several other contexts.

ECO-system includes all detailed placement methods implemented in Capo [107, 115, 116, 120], and can similarly be grafted onto other top-down placers, such as BonnPlace [138], PolarBear [50] or NTUPlace [74], by performing a one-pass Fiduccia-Mattheyses test. ECO-system can act like the WSA technique [93], and can invoke any black-box global placement algorithm when it decides that a particular bin must be replaced from scratch.

The definitive success of ECO-system in legalizing APlace and mPL6 global placements (Tables 6.2, 6.4, 6.5 and 6.6) allows one to answer a long-standing question in placement — whether the slicing structure of min-cut placements costs them HPWL. Given that the placements produced by ECO-system are slicing, the answer is negative for standard-cell placement, but is likely to be positive when large macros are involved, as suggested by results in [107].

We have analyzed requirements for an ECO placement tool and implemented an interface based on ECO-system applicable to high-level and physical synthesis, allowing the designer to add and remove nets and cells from a design, reallocate whitespace and large macros (Figure 6.1), resize cells and re-weight nets while retaining control of the amount of change performed by ECO-system.

6.6.2 Additional Applications

As ECO-system subsumes and generalizes the WSA technique [92, 93] and outperforms the technique from [5], ECO-system can also be used for the applications studied in previous publications. In addition to our experiments that demonstrate improvement in routability and support for gate sizing, ECO-system can be used to support buffer insertion in physical synthesis and floorplan resizing during chip planning [92].

Another relevant application of ECO-system lies in fault-tolerant reconfigurable computing. In this paradigm, the digital system periodically invokes built-in self-test and may identify components that recently failed. To avoid using faulty components, the system can be reconfigured to use only those resources that remain operational.

ECO-system could be used to quickly reprogram faulty chips in the following way.

Obstacles are placed in those areas of a circuit that have been determined to have errors. ECO-system can be run on this modified design to remove all overlaps between the old placement and the new fixed objects. The legalized placement would then be free of errors as none of the faulty parts would be used in the replacement. ECO-system uses as much of the original placement as possible so timing and other relevant circuit properties would likely be preserved.

Algorithms used in ECO-system can also be used to geometrically partition a layout so as to minimize interconnect between partitions, as shown in Figure 6.3. With minimal communication between partitions, physical design algorithms that are generally run after placement (such as cell sizing, routing or buffer insertion) can be parallelized, improving runtime on multi-processor systems. In particular, it has been shown that post-placement optimizations for timing can be parallelized [87]. Empirical results show that runtime can be decreased by up to 5x when running on a parallel machine with eight processors [87].

6.6.3 Our Contributions to Shared Research Infrastructure

All algorithms reported in this work are now available to the research community in source code form, integrated into the Capo placer — an established open-source software distribution. The availability of ECO-system in this work significantly lowers barriers for entry in two research directions: (a) global placement and (b) physical synthesis. Indeed, work in global placement has always been complicated by the need to produce legal, routable placements, but with the availability of a fast and reliable legalizer it becomes easy to evaluate new global placement techniques without a significant infrastructure investment. Similarly, our software allows one to experiment with physical synthesis opti-

mizations (e.g., sizing, buffering) and placement-driven logic transformations (e.g., fanout optimization) while delegating legalization to ECO-system.

PART III

Fundamental Techniques for Routing

CHAPTER VII

Our Framework for Global Routing

In this chapter we develop a high-performance routing technique based on *Discrete Lagrange Multipliers* (DLM). In addition to its strong empirical performance, DLM offers a natural way to handle net weights and timing optimization in routing, and explains several empirical effects observed in negotiated-congestion routing techniques such as the *last-gasp problem* and the relative simplicity of two-dimensional formulations compared to multi-layer (three-dimensional) formulations. Proposed algorithms are implemented in FGR[1], a high-performance global router for nanometer scale designs.

Our key contributions are as follows:

- A routing technique based on Discrete Lagrange Multipliers (DLM) which provides a natural way to handle net weights and timing optimization in global routing. FGR

[1] "A Fairly Good Router"

handles two- and three-dimensional routing of ASICs with millions of nets. This is almost an order of magnitude greater than what has been reported in the literature for most ASIC and FPGA routers. In the 32-bit address space, FGR scales up to 1,000,000 nets.

- Extensions of A*-search to restructure net topologies so as to avoid congestion and circumvent obstacles.
- Improved wirelength on the ISPD '07 Global Routing Contest suite [71]. FGR produces smaller wirelengths than the winners of the contest *on every benchmark*, and is able to route without overflows every benchmark that the winners routed without overflows. In terms of wirelength, FGR outperforms BoxRouter [40] by 9.5% and MaizeRouter [99] by 8.0%.
- Violation-free routing of all ISPD '98 IBM benchmarks [70], unlike routers in previous literature. FGR uses 35% less runtime than BoxRouter and produces solutions with 2.7% smaller wirelength.
- Thorough empirical evaluation of several routing strategies and algorithms including net decomposition by MST vs. Steiner trees and layer assignment for three-dimensional routing problems vs. direct three-dimensional maze routing. We identify previously unreported bottlenecks, such as the *last-gasp* problem in negotiated-congestion routing, and propose solutions.

This chapter is organized as follows. In Section 7.1 we describe the architecture of the FGR router, the mathematical basis for its key algorithms, and important insights into the integration of major components. We benchmark FGR against state of the art in Section 7.2 and conclude in Section 7.3.

7.1 High-performance Global Routing

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

7.1.1 Basic Algorithmic Framework

Routing algorithms must carefully balance wirelength minimization and congestion. Some detours may be necessary to avoid routing violations and overcapacity GCells, but excessive detouring leads to over consumption of routing resources, aggravating congestion. In particular, the results of the ISPD '07 routing contest [71] show that some routers are good at finding violation-free solutions, some are good at minimizing wirelength, but few are good at both. This trend is illustrated in Figure 3.5 which shows routed wirelength vs. violation count for two-dimensional solutions submitted to the contest. A likely source of this inflexibility is the common use of uniform, predetermined rules in all regions of the chip as in FastRoute [109, 110] and the Chi dispersion router [64].

In continuous optimization, dynamic pricing of constraint satisfaction can be modeled by Lagrange multipliers — a mathematical method for optimizing a multivariate function subject to a number of constraints [89]:

$$\begin{gathered} \min_{\mathbf{x}\in X} W(\mathbf{x}) \\ \text{subject to } C_e(\mathbf{x}) = 0, \;\; 1 \leq e \leq n \end{gathered} \tag{7.1}$$

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

$$F(\mathbf{x}, \lambda) = W(\mathbf{x}) + \sum_{e=1}^{n} \lambda_e C_e(\mathbf{x}) \tag{7.2}$$

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

$$W(\mathbf{x}) = \sum_{i=1}^{m} R_i(\mathbf{x}) = \sum_{e=1}^{n} B_e(\mathbf{x}) = \sum_{e=1}^{n} \left(\sum_{\text{net } i \text{ uses } e} b_e \right) \tag{7.3}$$

where $R_i(\mathbf{x})$ is the number of segments used by net i and $B_e(\mathbf{x})$ is the number of nets using edge e. Thus (7.2) can be rewritten

$$F(\mathbf{x}, \lambda) = \sum_{e=1}^{n} \left(B_e(\mathbf{x}) + \lambda_e C_e(\mathbf{x}) \right) \tag{7.4}$$

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

$$\lambda^{k+1} = \lambda^k + \alpha C(\mathbf{x}^k) \tag{7.5}$$

where $\alpha > 0$ is a line-search parameter. Note the similarity in the update of the Lagrange multipliers and how h_e is updated in Formula 3.2. While we are also dealing with large sparse problems, they are discrete and non-convex. This calls for a different iterative optimization procedure, such as *greedy search*, *hill-climbing* or *rip-up-and-re-route*. However, since Lagrange multipliers remain continuous, the same update rule can be adopted.[2]

Interpreting Formula 7.4 for a given net i in terms of NCR yields

$$c_e = b_e + h_e \cdot p_e \tag{7.6}$$

[2]To this end, the use of Lagrange multipliers can be viewed as a way to leverage continuous optimization in a discrete domain, such as nanoscale routing.

which is different than Formula 3.1 [98], but also makes more sense since it preserves the base cost. Therefore FGR uses this Discrete Lagrange Multiplier (DLM) formulation instead of NCR which was used in FGR's ISPD '07 contest submission. To compute p_e, we use a new penalty function introduced in Section 7.1.2 below. Furthermore, the justification of dynamic cost updates through DLMs explains the results we see in Sections 7.1.4, 7.1.5 and 7.2.

In addition to being a rigorous mathematical technique, the use of Lagrange multipliers often admits application-specific interpretation. For example, it is used in macroeconomics to mathematically describe market pricing — in a market economy, adequate resource pricing encourages consumers to look for competitive alternatives, leaving the most expensive resources to the consumers that gain most. A very similar interpretation holds in the case of routing, and the "fairness" of this pricing system is confirmed by good convergence properties in practice, as illustrated in Figures 7.5 and 7.6.

In the initial routing formulation (Equation 7.1) all nets are treated equally when optimizing total wirelength, but in many cases certain nets are more important than others for optimization, as in timing-driven routing. Each net is assigned a weight, and the goal is to optimize total weighted wirelength. Weighted wirelength is written as

$$W'(\mathbf{x}) = \sum_{i=1}^{m} w_i R_i(\mathbf{x}) = \sum_{e=1}^{n} B'_e(\mathbf{x}) \tag{7.7}$$

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

$$B'_e(\mathbf{x}) = \sum_{\text{net } i \text{ uses } e} w_i \cdot b_e \tag{7.8}$$

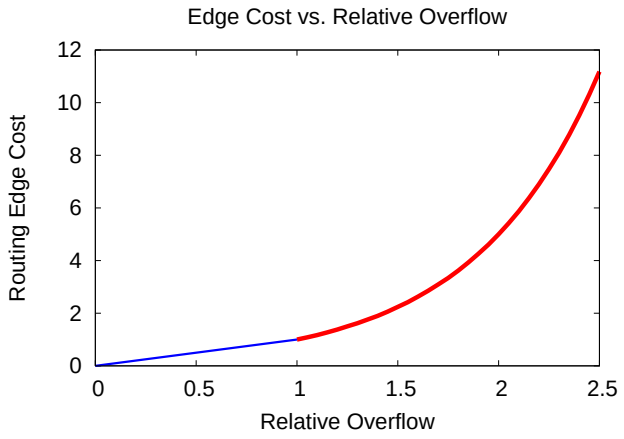


Figure 7.1: Cost of a routing edge as a function of relative overflow. Cost is linear while the edge is not overfilled, but grows exponentially once the edge is overfull.

By replacing B_e in Formula 7.4 with B'_e, we get

$$F(\mathbf{x}, \lambda) = \sum_{e=1}^{n} \left(\left(\sum_{\text{net } i \text{ uses } e} w_i \cdot b_e \right) + \lambda_e C_e(\mathbf{x}) \right) \tag{7.9}$$

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

$$c_e(i) = w_i \cdot b_e + h_e \cdot p_e \tag{7.10}$$

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

7.1.2 Congestion Penalty

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

$$p_e = \begin{cases} \exp\left(k(\omega_e - 1)\right) & \text{if } \omega_e > 1 \\ \omega_e & \text{otherwise} \end{cases} \tag{7.11}$$

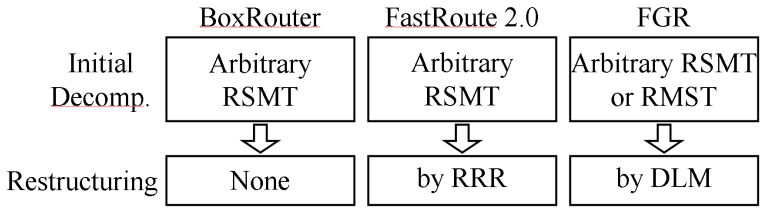


Figure 7.2: A comparison of the net decomposition techniques used by BoxRouter [40], FastRoute 2.0 [110] and FGR. In Section 7.2.2, we compare the use of RMSTs and RSMTs in FGR.

The exponential nature of our cost function for overfull routing edges serves to amplify congestion and gives the maze router incentive to avoid overfull edges when re-routing nets (see Figure 7.1, where $k = \ln 5$). We have studied $0 < k \leq \ln 10$ and found that higher values of k reduce runtime, but increase detouring and routed length. FGR uses $k = \ln 5$ by default. Instead of using uniform weights of 1 for routing edges to create an initial routing solution, which is common in NCR, FGR uses $b_e + p_e$ as the weight for edges to create an initial solution, where p_e is calculated according to Equation 7.11.

7.1.3 Interactions Between Single- and Multi-Net Routing

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

Figure 7.2 compares the net decomposition and restructuring techniques used by FGR to those in prior work. During DLM, the most congested subnets are ripped up and rerouted by A*-search. When ripping up a subnet with endpoints P_1 and P_2, FastRoute 2.0 tries to reconnect the two components of the net, not necessarily using P_1 or P_2, which invalidates the point-to-point lower bound used in A*-search. When re-routing a subnet,

FGR requires the replacement segments to pass between P_1 and P_2, based on the following result.

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

Proof: Assume there is a shortest path $A \rightarrow B$ joining T_1 and T_2 such that $A \in T_1$ and $B \in T_2$. As T_1 and T_2 are trees, there exist unique non-self-intersecting paths $P_1 \rightarrow A$ and $B \rightarrow P_2$ using only edges contained in T_1 and T_2, respectively. As the costs of tree segments are zero, $cost(P_1 \rightarrow A) = cost(B \rightarrow P_2) = 0$. Thus $cost(P_1 \rightarrow A \rightarrow B \rightarrow P_2) = cost(A \rightarrow B)$. For the sake of contradiction, assume that $P_1 \rightarrow A \rightarrow B \rightarrow P_2$ is not a shortest path; there exists[3] path $P_1 \rightsquigarrow P_2$ with $cost(P_1 \rightsquigarrow P_2) < cost(A \rightarrow B)$. But $P_1 \rightsquigarrow P_2$ connects T_1 and T_2, so $cost(P_1 \rightsquigarrow P_2) \geq cost(A \rightarrow B)$. Contradiction.

Conversely, let $P_1 \rightarrow P_2$ be a shortest path. Let C be the last vertex along $P_1 \rightarrow P_2$ such that $C \in T_1$ and let D be the first vertex along $P_1 \rightarrow P_2$ such that $D \in T_2$. As T_1 and T_2 are trees, there exist unique non-self-intersecting paths $P_1 \rightarrow C$ and $D \rightarrow P_2$ using only edges contained in T_1 and T_2, respectively. $cost(P_1 \rightarrow C) = cost(D \rightarrow P_2) = 0 \Rightarrow cost(P_1 \rightarrow P_2) = cost(C \rightarrow D)$. Assume for the sake of contradiction $C \rightarrow D$ is not a shortest path for T_1 and T_2; there exists path $A \rightsquigarrow B$, $A \in T_1, B \in T_2$, with $cost(A \rightsquigarrow B) < cost(C \rightarrow D) = cost(P_1 \rightarrow P_2)$. There exist $P_1 \rightsquigarrow A$ and $B \rightsquigarrow P_2$ such

[3] In this proof, $\rightsquigarrow$ denotes paths assumed to exist for the sake of contradiction.

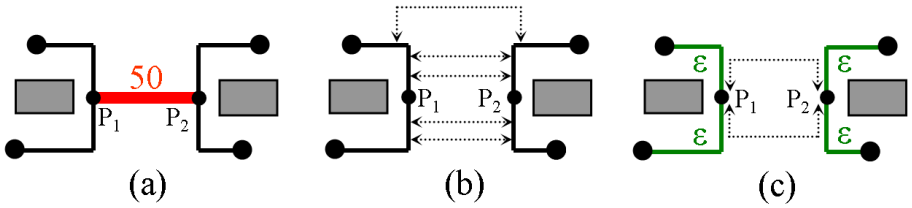


Figure 7.3: Re-routing a subnet and changing net topology in FGR. The shaded boxes represent obstacles. The tree in (a) passes through a congested segment in the middle which must be ripped up. The dashed arrows in (b) represent several possible re-routings that a restructuring algorithm may consider. The re-routings shown in (c) are two that FGR will consider during DLM. Paths considered by FGR must start and end along the endpoints of the segment that was removed. Both of these re-routings reuse routing segments from the net and create new Steiner points if chosen. The use of temporary zero-cost edges is required to preserve the efficiency of A*-search.

that $cost(P_1 \rightsquigarrow A) = cost(B \rightsquigarrow P_2) = 0 \Rightarrow cost(P_1 \rightsquigarrow A \rightsquigarrow B \rightsquigarrow P_2) = cost(A \rightsquigarrow B) < cost(P_1 \rightarrow P_2)$. Contradiction. □

Temporary change of edge costs to 0 is easy to implement during A*-search because we route one net at a time and can undo any cost adjustments before considering other nets. However, in order to use A*-search, we must supply a correct lower bound. We normally use the three-dimensional Manhattan distance multiplied by the minimum cost of any routing segment. The naive solution — to ignore the 0-cost edges — may produce estimates that are greater than the true cost, which would invalidate A*-search. However, if we literally set an edge's cost to zero, the lower bound will automatically become zero. Therefore, in our implementation we set the cost of previously used edges to $\varepsilon > 0$, a very small value. This technique is illustrated in Figure 7.3, where FGR modifies the net topology to avoid congestion.

While prior state-of-the-art routers (BoxRouter, FastRoute and MaizeRouter) consistently start by decomposing multi-pin nets with minimal Steiner trees, we believe that our

ALGORITHM 7.1: Layer assignment

```
    ▷ Input: two-dimensional routing solution, 2dsol
    ▷ Output: three-dimensional routing solution, 3dsol
 1  foreach(net n in 2dsol)
 2      do foreach(subnet s of n)
 3          do route ← GETROUTE(s)
 4             currPoint ← GETSTARTTERMINAL(s)
 5             currLayer ← GETLAYER(currPoint)
 6             while (currPoint ≠ GETENDTERMINAL(s))
 7                 do nextPoint ← GETNEXTPOINT(route, currPoint)
 8                    nextLayer ← the layer closest to currLayer where adding an
                      edge connecting currPoint and nextPoint causes least overflow
 9                    Add a segment from currPoint to nextPoint on layer nextLayer to 3dsol
10                    Add vias connecting (currPoint.x,currPoint.y,currLayer) and
                      (currPoint.x,currPoint.y,nextLayer) to 3dsol
11                    currPoint ← nextPoint
12                    currLayer ← nextLayer
13             Add vias connecting (currPoint.x,currPoint.y,currLayer) and
               (currPoint.x,currPoint.y,GETLAYER(GETENDTERMINAL(s))) to 3dsol
```

Figure 7.4: Layer assignment in FGR.

integration of topology restructuring into a powerful DLM framework facilitates additional opportunities. As illustrated in Figure 3.2, Steiner trees tend to generate net decompositions with many flat subnets which offer no flexibility in routing. MSTs tend to have fewer edges but with more flexibility, which can be exploited by DLM to avoid congestion. Moreover, the gradual addition of sharing to MSTs during DLM-based topology restructuring can generate high-quality congestion-driven Steiner trees without the need to estimate congestion before routing. Starting with minimal Steiner trees seems to require heavier restructuring to achieve similar effects, and could not only slow down maze routing, but also make RRR or DLM less successful. Using RSMTs vs. RMSTs is covered in Section 7.2.2.

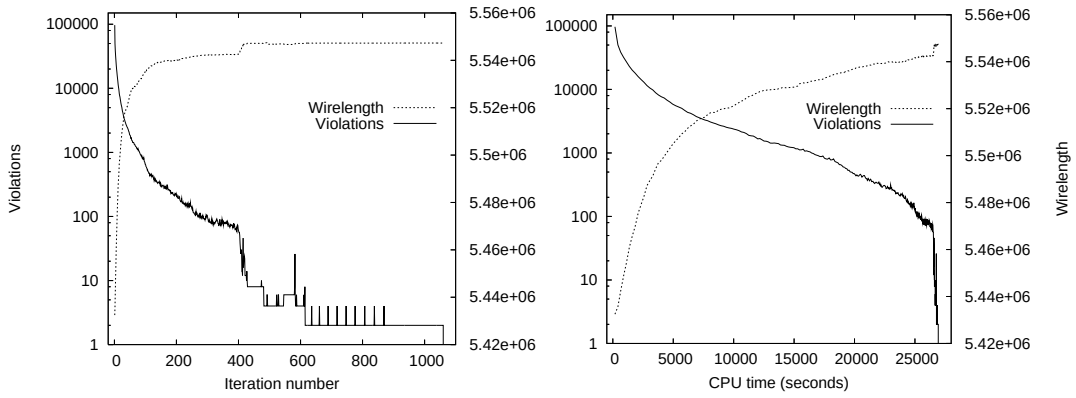
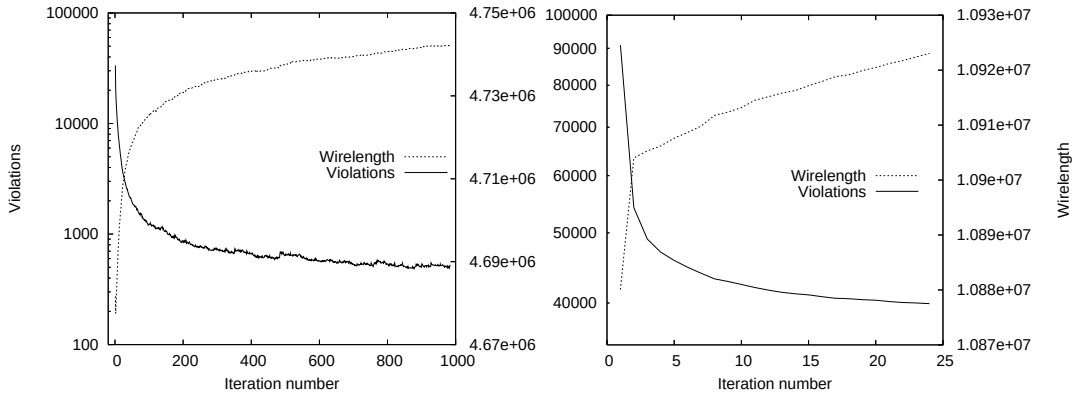


Figure 7.5: Violation count and wirelength on the two-dimensional ISPD '07 benchmark `adaptec1` plotted as a function of (a) iteration number and (b) time. Violation counts are plotted on a log-scale and decrease, while wirelength is plotted on a linear scale and monotonically increases. Note that the majority of DLM iterations occur when 100 or fewer violations remain, but total wirelength noticeably increases during that phase.

7.1.4 Overcoming the "Last-gasp" Problem

Discrete Lagrange multipliers work well at the large scale because the statistical behavior of numerous discrete variables is not very different from the continuous case. However, when only several violations remain, the routing task becomes much more discrete. In our experiments with almost every benchmark we have observed unusual behavior where FGR spends many DLM iterations when its solution is nearly legal before it is able to terminate with a completely legal solution. Indeed, more than 75% of DLM's iterations for the `adaptec2` benchmark [71] take place when less than 0.01% of routing segments have overflow (see also Table 7.4). We term this undesirable behavior the *last-gasp* problem and illustrate it on the `adaptec1` two-dimensional benchmark in Figure 7.5. To rectify this situation, we propose the following improvement. When the percentage of routing edges with overflow becomes small, we restrict the maze router to using only edges that have available space and weigh routing edges only by their base cost b_e. Thus if there is



(a) `newblue1` (331663 nets) (b) `newblue3` (551667 nets)

Figure 7.6: Violation count and wirelength plotted as a function of iteration number on two unroutable two-dimensional ISPD '07 benchmarks. In both cases, FGR is stopped after a period of 24 hours.

any way to route the net without causing overflow, we will take it to avoid further rip-up iterations. Otherwise, default DLM is used. In many cases this last phase of DLM reduces iterations without impacting total routed wirelength.

7.1.5 Three-dimensional Routing

The difficulties experienced by DLM due to discreteness also suggest that traditional two-dimensional routing may be considerably easier than proper three-dimensional routing where smaller edge capacities are spread through multiple routing layers. In other words, aggregating edge capacities in one layer would encourage continuous-like resource pricing, making it easier to satisfy all constraints. This is consistent with observations from experiments shown in Section 7.2.3.

FGR performs three-dimensional routing by first projecting the routing instance onto a two-dimensional grid and aggregating the capacities of edges that project onto each other. This grid contains a single layer of horizontal wires and a single layer of vertical wires connected by a layer of vias, such as grid depicted at the right of Figure 3.1. Capacities

on higher layers may be smaller due to increased pitch, but for each routing grid edge we calculate the number of wires that are allowed to pass through it, which takes wire widths and pitches into account. FGR routes this two-dimensional problem instance as normal until a legal solution is found or a runtime/iteration limit is reached. Next FGR performs layer assignment for each routing segment used in the two-dimensional solution.

Theorem 2 *If the projected two-dimensional instance has a legal solution and via counts are unconstrained, then the original three-dimensional instance must have a legal solution.*

Proof: Three-dimensional routes can be constructed by the algorithm in Figure 7.4. □

FGR's method will produce a three-dimensional solution that uses exactly the same number of routing segments as the two-dimensional solution, but differs in via counts. Unfortunately the difference in via counts is usually large and proportional to the number of layers in the three-dimensional instance. To counteract this phenomenon, we perform a single round of RRR for every subnet to reduce vias. In this round of optimization, the cost of each routing segment is much simpler than in DLM: each routing segment is assigned a cost of 1 and vias are priced as in Section 7.1.6. It is easy to lower-bound the cost of a path with these edge costs by the three-dimensional Manhattan distance, so it is particularly amenable to A*-search. Each subnet is ripped up and rerouted by the maze router individually, and edges with no spare capacity are not allowed. While Theorem 2 is not a surprising result, the fact that direct three-dimensional routing is less successful than two-dimensional routing with three-dimensional post-processing was unexpected and, in fact, undermined FGR's performance in the ISPD '07 routing contest.

Table 7.1: Routed cost breakdown of FGR's solutions to the ISPD '07 Global Routing Contest benchmarks [71]. "FLUTE Ratio" is the ratio of the length of routing segments used to the Steiner tree length of all nets as computed by FLUTE [44]. Vias account for more than 25% of total cost in every two-dimensional benchmark and more than 50% of total cost in each three-dimensional benchmark, highlighting the importance of via minimization.

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

7.1.6 Via Pricing and Optimization

Given that the resistivity of tungsten (the material of vias) is much higher than that of copper and aluminum, vias are critical in timing-driven routing. The high variability in via parasitics [121] and the common practice of post-route via doubling to improve yield [91, 94] suggest that via minimization is a key issue in routing at the nanometer scale. Furthermore, an unnecessarily large number of vias can hamper routability because each via obstructs a section of its track. Table 7.1 illustrates just how significant vias are in the ISPD '07 contest benchmarks. Vias represent from 26% to 49% of the total cost of FGR's solutions to the two-dimensional benchmarks. Comparing two-layer routing with 6-layer routing, via counts approximately triple and account for 50% to 74% of total cost.

Table 7.2: Statistics of the ISPD '98 IBM benchmark suite [70]. Runtimes for BoxRouter [40] and FGR are given in seconds. FGR is faster than BoxRouter on 7 of the 10 benchmarks and uses 35% less runtime to solve the entire suite.

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

The routing framework closest to ours — Negotiated-Congestion Routing — does not consider via minimization because its focus is FPGA routing. To model the cost of vias, FGR treats them as segments in the routing graph. These segments connect adjacent routing layers as shown in Figure 3.1 and have unlimited capacity. Via routing segments have a different base cost, usually higher than that for regular segments. This flexibility allows FGR to price vias in specific applications. For example, in the ISPD '07 contest one via is equivalent to three routing grid segments, so the cost of vias in FGR is set to $3b_e$.

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

Table 7.3: Comparison of FGR to FastRoute 2.0 [110] and BoxRouter [40] on the ISPD '98 IBM benchmark suite [70]. FGR completes all 10 of the benchmarks while BoxRouter and FastRoute 2.0 leave overflow on 4 and 3 of the benchmarks, respectively. In terms of routed wirelength, FGR outperforms BoxRouter by 2.7% and FastRoute 2.0 by 3.6%.

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

7.2 Experimental Results

We have implemented FGR in C++ without external libraries (compiled with GCC 3.4.5), but added optional interface to the Steiner-tree packages FLUTE [44] and Fast-Steiner [78] to compare them with MST decompositions. The core algorithms and data structures of FGR were implemented in one month. All runs were performed on 2.4 GHz Opteron workstations running Linux. FGR was compiled in 32-bit mode and was therefore limited to less than 4GB of RAM.

7.2.1 Performance on ISPD '98 and '07 Benchmarks

Table 7.2 describes the ISPD '98 IBM benchmarks and compares FGR to BoxRouter [40] in terms of runtime. Table 7.3 compares FGR to BoxRouter and FastRoute 2.0 [110] in terms of solution quality. Unlike all previous routers in the literature, FGR is able to route all of the IBM designs without overflow. Both BoxRouter and FastRoute 2.0, which report the best results on this suite so far, produce solutions with overflow on 4 and 3 of the

Table 7.4: Statistics of the ISPD '07 Global Routing Contest benchmarks [71]. For FGR we list runtime (in minutes), the number of iterations of rip-up-and-re-route (which are very similar for two- and three-dimensional variants), and maximum memory usage, which is significantly greater for three-dimensional than for two-dimensional variants.

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

benchmarks, respectively. Overall, FGR produces solutions with 2.72% less wirelength than BoxRouter and 3.62% less wirelength than FastRoute 2.0. In addition, FGR is faster than BoxRouter on 7 of the 10 benchmarks and uses 35% less runtime to complete the entire suite. Unlike the ISPD '07 contest benchmarks, the ISPD '98 benchmarks feature only a single metal layer, making via minimization unnecessary.

Table 7.4 shows statistics of the benchmarks used at the ISPD '07 Global Routing Contest [71]. These benchmarks are considerably larger than the ISPD '98 benchmarks and include both two- and three-dimensional variants. These benchmarks also feature non-trivial routing obstacles, and, consequently, routing resources are not spread evenly throughout the layout as in the ISPD '98 suite. Table 7.4 also shows runtimes and memory requirements for FGR on these benchmarks. In all cases FGR stays within the 32-bit memory space and finishes well under a given 24-hour timeout on all but the `newblue1` and `newblue3` benchmarks on which no router at the ISPD '07 contest was able to find a legal solution.[4]

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

Table 7.5: Comparison of FGR to the other top-three routers at the ISPD '07 Global Routing Contest [71]. FGR routes as many benchmarks without overflow as the winners of the contest with 7.0% better wirelength than the best of BoxRouter [40] and MaizeRouter [99]. *The `adaptec4` three-dimensional and `newblue2` three-dimensional benchmarks were routed using FGR's option "-full3d".

Bench-mark		Best of BoxRouter and MaizeRouter				FGR			
		Overflow		Cost	Router	Overflow		Cost	vs.
		total	max	(e5)		total	max	(e5)	Best
adaptec	#1 2-d	0	0	58.84	Box	0	0	**54.44**	-7.48%
	#1 3-d	0	0	99.61	Maize	0	0	**88.45**	-11.20%
	#2 2-d	0	0	55.69	Box	0	0	**52.30**	-6.09%
	#2 3-d	0	0	98.12	Maize	0	0	**89.89**	-8.39%
	#3 2-d	0	0	137.75	Maize	0	0	**130.89**	-4.98%
	#3 3-d	0	0	214.08	Maize	0	0	**199.66**	-6.74%
	#4 2-d	0	0	128.45	Maize	0	0	**125.00**	-2.69%
	#4 3-d	0	0	194.38	Maize	0	0	**179.36***	-7.73%
	#5 2-d	0	0	164.32	Box	0	0	**152.13**	-7.42%
	#5 3-d	0	0	298.08	Box	0	0	**259.98**	-12.78%
newblue	#1 2-d	400	2	51.13	Box	526	4	**47.42**	-7.26%
	#1 3-d	400	2	101.83	Box	514	2	**94.26**	-7.43%
	#2 2-d	0	0	79.64	Maize	0	0	**76.51**	-3.93%
	#2 3-d	0	0	139.66	Maize	0	0	**129.40***	-7.35%
	#3 2-d	32588	1236	114.63	Maize	39908	1120	**109.23**	-4.71%
	#3 3-d	32840	1058	184.40	Maize	39828	374	**173.71**	-5.80%
Average									**-7.03%**

Next, we compare FGR to the routers that scored best at the ISPD '07 contest. Since an earlier version of FGR placed 1st in the two-dimensional category, we exclude it from comparison (however, the version we report improves upon FGR's results in the contest on every benchmark). We compare FGR to MaizeRouter [99] which placed 1st in three-dimensions and 2nd in two-dimensions, and to BoxRouter which placed 2nd in three-dimensions and 3rd in two-dimensions. FGR produces smallest wirelengths *on every benchmark* and is able to route without overflow every benchmark that was legally routed at the contest. In particular, FGR outperforms BoxRouter in wirelength by 9.5% and MaizeRouter by 8.0%.

Table 7.6: Comparing net decomposition by MST versus Steiner trees on the ISPD '07 benchmarks [71]. Time taken for decomposition by MST or Steiner trees is less than 1 minute on all benchmarks. While using Steiner tree decompositions results in a reduction in routed segment length of 0.5%, it increases via counts by 1.8% and thus increases the total cost of routing solutions by 0.7%. Decomposition by Steiner trees increases routing time by 22%.

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

7.2.2 Using Steiner Trees versus Using MSTs

Traditionally net decomposition has been done using Minimal Spanning Tree (MST) algorithms, but fast and extremely accurate Rectilinear Steiner Minimal Tree (RSMT) construction algorithms have become increasingly popular in the literature [40,109,110]. FGR can use any well-formed net decomposition, so we study how the choice of net decomposition affects FGR's overall results—we compare MST to a combination of FLUTE [44] and FastSteiner [78] that returns the better Steiner tree every time. FGR merges segments of decomposed nets, as described in Section 7.1.3 and produces non-trivial Steiner trees even when given decompositions by MSTs. The results on the ISPD '07 benchmarks are shown in Table 7.6. Time taken for decomposition by MSTs or Steiner trees is less than 1

Table 7.7: Comparing layer assignment with full three-dimensional routing on the three-dimensional instances of the ISPD '07 benchmarks [71]. Total cost of the better solution (compared first by overflow then total cost) for each benchmark is highlighted.

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

minute on all benchmarks and does not significantly impact runtimes. As expected, routed segment length is smaller when Steiner tree algorithms are used. On the other hand, using Steiner tree algorithms actually increases via counts by 1.8% and causes total cost to increase by 0.7%. All evidence we have seen suggests that MST decompositions leave more flexibility than minimum Steiner trees, allowing one to avoid some amount of detouring. Prior work has shown that optimal Steiner trees for a given set of points can vary widely, but specialized techniques can increase flexibility [15]. However, FLUTE and FastSteiner do not currently optimize tree flexibility. In addition, Steiner points may inadvertently be placed in congested areas by the Steiner tree constructor, causing increased congestion and detouring. Congestion-driven Steiner trees could be helpful in this context, but apparently MSTs already provide a good solution and can also be biased to avoid congestion.

7.2.3 Layer Assignment versus Full Three-dimensional Routing

In section 7.1.5 above we described that FGR performs three-dimensional routing by first flattening the routing instance onto a two-dimensional grid, routing the new two-

dimensional problem instance, and then converting the two-dimensional solution into a three-dimensional solution by assigning layers to routed segments, adding vias as necessary. FGR is also capable of solving three-dimensional problems directly by using full three-dimensional maze routing, and in Table 7.7 we compare both methods. It is readily apparent that full three-dimensional routing takes far longer than two-dimensional routing with layer assignment, most likely because three-dimensional routing is more complex. On the easiest benchmarks, `adaptec4` and `newblue2`, full three-dimensional routing takes at least 50% longer, but is able to decrease via counts significantly and in turn improve total cost by 2.0% and 2.1%, respectively. On the other hand, on the benchmarks where FGR with layer assignment cannot find a legal solution within 24 hours, `newblue1` and `newblue3`, full three-dimensional routing produces solutions with significantly more overflow.

7.2.4 Selective Net Weighting

To avoid detouring critical nets, identified outside of the router by their timing criticality, FGR can route them preferentially by assigning net weights and minimizing weighted wirelength. To validate this method, we route the `newblue2` benchmark from the ISPD '07 contest. We choose a random subset of 10% of the nets of the design, double their weight, and route from scratch. Distributions of detours on the nets are shown in Figure 7.7. Detouring on the nets with higher weight is reduced as is the overall detouring on the design. Runtime and total wirelength are affected negligibly. Thus using net weights is an effective method for controlling detouring and timing on selected nets.

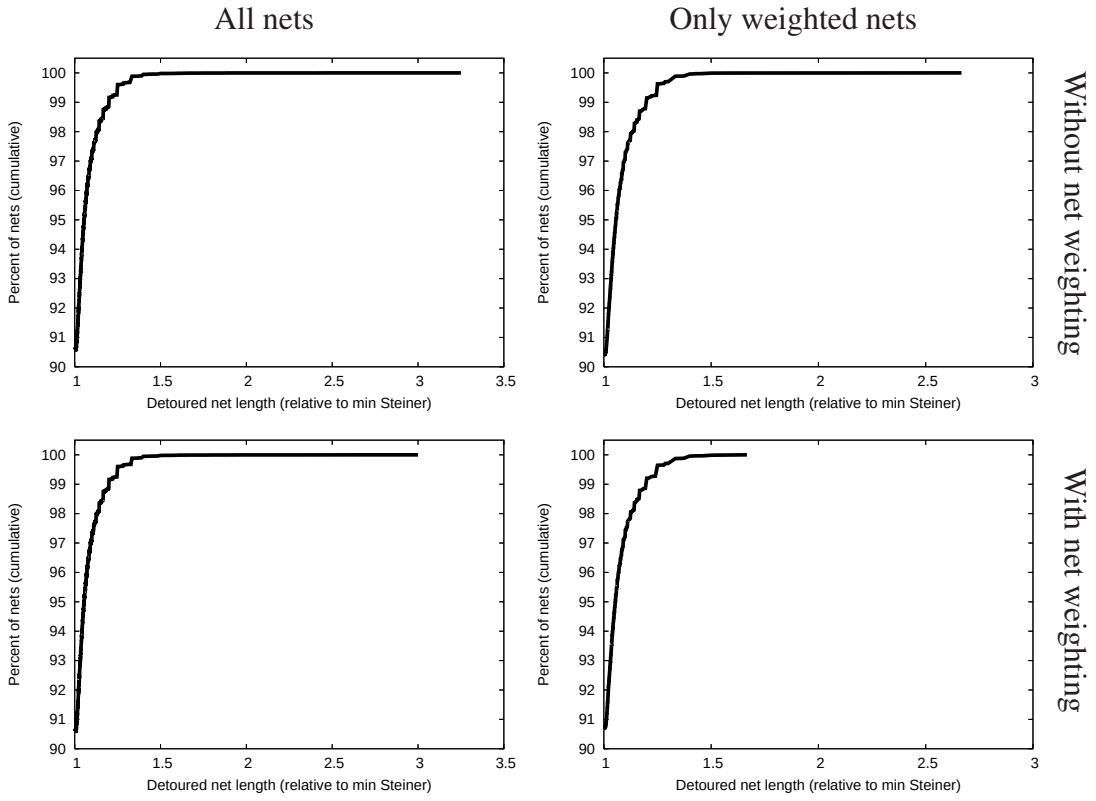


Figure 7.7: Cumulative distributions of detouring without (above) and with (below) net weighting on the two-dimensional `newblue2` benchmark. Net detours are measured as a ratio of routed net length to Steiner wirelength as given by FLUTE [44]. When weights are applied to a subset of the nets, the detouring on those nets goes down significantly without adverse effects on the detouring of all nets.

7.3 Conclusions

In this chapter we have presented FGR, a high-performance global router for nanometer scale designs. FGR's implementation is very compact—core algorithms and data structures require only 1200 lines of C++ code. FGR outperforms the best results from the ISPD '07 Global Routing Contest, as well as previous literature, in terms of route completion, runtime and total wirelength. In particular, FGR improves upon wirelengths produced by BoxRouter and MaizeRouter in March 2007 by 9.5% and 8.0%, respectively.

CHAPTER VIII

Extensions to Our Routing Framework

The ISPD 2007 Global Routing Contest organized by IBM Austin Research Laboratory changed the landscape of global routing research by introducing sixteen new benchmarks that are orders of magnitude more challenging than previously available layout instances [71]. These design examples stimulated the development of new algorithms and software, leading to improved performance and robustness of state-of-the-art tools. For example, before the contest no academic global router had been able to completely route the ISPD'98 suite of benchmarks. Yet, three routers presented at ICCAD'07 – Archer [108], BoxRouter 2.0 [42] and FGR – legally route these benchmarks in a matter of seconds. Two of them successfully competed in the ISPD'07 contest.

According to a recent survey by Lou Scheffer [122, Chapter 8], industrial routers most often rely on fairly basic techniques, such as maze routing and rip-up and reroute, rather than methods that make use of "higher mathematics." At each iteration of rip-up and reroute, routes that pass through congested regions are first removed, and then greedily routed one by one, such that the order is different every time. The iterations stop when a completely legal solution is found or a timeout is reached. In contrast to this simple but

effective technique, most of pre-ISPD'07 academic research had been focused on highly sophisticated combinatorial techniques such as multi-commodity flows (MCF) and integer linear programming (ILP) to route nets simultaneously. Somewhat in agreement with [122, Chapter 8], the ISPD'07 contest was won by two routers (FGR and MaizeRouter) that did not use these paradigms. Moreover, a recent publication [42] describing the third best-performing router from the contest, BoxRouter 2.0, showed that it performs much of its work using rip-up and reroute.[1] This is a significant change from BoxRouter 1.0 that relied heavily on ILP. Archer [108], presented at ICCAD'07 reports low runtime and competitive results for two-dimensional routing. However, it does not achieve best wirelength on any of the benchmarks, and lags behind FGR and BoxRouter 2.0 dramatically in multi-layer routing. Academic research has clearly shifted in 2007 as all three global routers presented at ICCAD'07 use modifications of Negotiated Congestion Routing (NCR), a rip-up and reroute technique introduced in the PathFinder FPGA router [98] but not used before in ASIC routing.

Recent publications [42,108,110] describe a variety of techniques, some new and some old, used in competitive routers. However, as we demonstrate in Sections 8.1-8.3, some of these techniques are superseded by others, and some appear unnecessary. To this end, a major challenge addressed by our work is to identify a minimal set of high-performance routing techniques that is sufficient to achieve best results in the "wirelength $\times$ runtime $\times$ violations" space.

In this chapter we introduce improvements to FGR which we collectively call Sherpa.

[1] The BoxRouter 2.0 work also does not list runtimes, indicating that they may be significant. Indeed, the authors of BoxRouter acknowledged at ICCAD'07 that BoxRouter 2.0 requires two days to route the `adaptec5` benchmark.

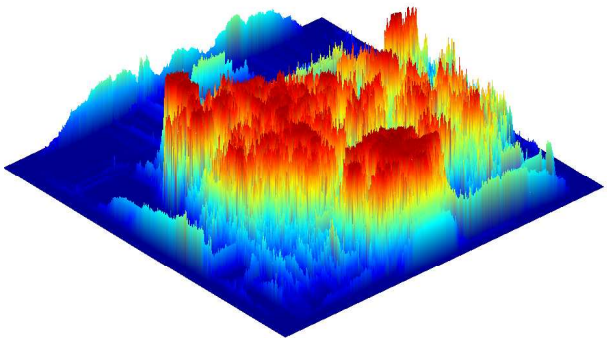


Figure 8.1: Congestion map of the `newblue1` two-dimensional benchmark as guided by Sherpa.

The contributions of this chapter are summarized as follows:

- An account of efficient data structures for global routing, including a new branch-free data structure for single nets.
- A comparative analysis of single-net routing techniques that identifies several methods that are powerful, comprehensive and easy to implement.
- An explanation of high violation counts produced by FGR on the `newblue3` benchmark, and a more effective, logarithmic penalty function (LPF).
- Several new ideas in global routing that improve speed and solutions. These include dynamic adjustment of Lagrange steps (DALS), and a cyclical net-locking (CNL) technique.
- Unparalleled performance on the ISPD'07 routing benchmarks. In particular, Sherpa outperforms Archer and FGR in speed, while matching the wirelengths achieved by FGR and improving upon those by Archer and BoxRouter 2.0.

8.1 Data Structures for Routing

High performance, especially on large routing instances, demands transparent, memory-efficient data structures. What to store and how is equally important compared to what *not*

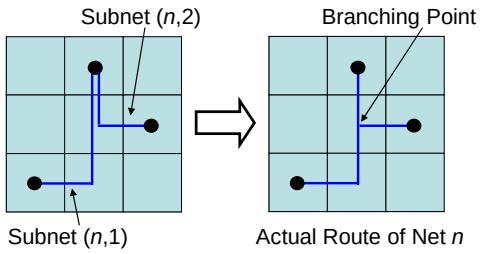


Figure 8.2: The branch-free representation (BFR) of routed nets. Subnets are treated separately and, when combined, form a completely routed solution without duplicate edges.

to store because excessive sophistication of data structures often leads to poor actual performance. Here we describe our basic data structures for individual routed nets and the dynamic global routing grid.

8.1.1 Branch-free Representation (BFR) for Individual Routed Nets

All global routers need to represent routes, and for nets with three or more pins there are several structural alternatives. The straightforward approach to this problem is to divide each net into a group of disjoint line segments, possibly with bends. In the case of 3-pin net n shown in Figure 8.2, this technique would add a branching point (Steiner point) to the middle of the net, creating three segments. This representation supports proper calculation of routing resources and is used in global routers such as FastRoute [110]. An additional recent requirement for routed net representations is flexibility—all competitive routers do some form of net restructuring whether it be during explicit steps such as in Archer [108] and BoxRouter [42], or continually through maze routing as in FastRoute and FGR. However, net restructuring steps cause branching points to move, appear, and disappear, which is difficult to support with this representation.

We take a different approach where branching points are represented implicitly. For

each *subnet*, a pair of real pins on a net, we store a list of the routing edges occupied by the subnet in addition to the coordinates of the endpoints of the subnet. These pairs must collectively form a spanning tree, e.g., a minimum spanning tree (MST). Each net lists the indices of routing edges it uses and allows one to find out how many subnets belonging to the net use a particular routing edge. Such a mapping can be implemented with an STL hash-map or balanced binary tree, but in practice they both require too much memory. Instead, our more memory-efficient data structure is an array of pairs of (1) routing edge indices and (2) the total number of subnets of the net that pass through the edge. Although this offers logarithmic look-up by routing edge and worst-case linear insertion and removal, in practice this consumes a trivial amount of runtime.

As each net stores the indices of used edges, routing resource usage can be calculated exactly and efficiently. These data structures also allow Sherpa to maintain Steiner-tree routings for nets without explicit representation of branching points. We call this a Branch-free Representation (BFR) for routed nets, and 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. In practice, overlap between subnets is typically small, and coalescing subnets automatically with BFR takes little time.

8.1.2 A Data Structure for Dynamic Global Routing Grid

The main challenges when designing a data structure for a routing grid are that the structure (i) is slim so as to improve cache locality (thereby reducing runtime) as well as fit large instances into the 32-bit address space, and (ii) provides constant-time access to 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 that tiles store their coordinates.

For each routing edge, we store: its type (VIA, HORIZONTAL or VERTICAL), the layer to which it belongs, the Lagrange multiplier associated with it (described in more detail in Section 8.3 below), routing resource capacity, current resource usage, and a list of the subnets that pass through it. Note that routing edges do not store additional information such as edge costs. There are two reasons for this. First, the functions we employ for determining edge costs can be computed quickly and on the fly 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.

8.1.3 Supporting Efficient Rip-up and Reroute

To facilitate efficient rip-up and reroute, fast identification of which subnets should be ripped-up at each iteration is crucial. Furthermore, the process of ripping-up a subnet must take negligible time in comparison to maze routing. In Section 8.1.2 above, recall that a routing edge maintains a list of the subnets that pass through it. Thus, to quickly determine which connections need to be adjusted during an iteration of rip-up and reroute, one iterates over all routing edges, determines which edges are over-capacity, and adds the subnets using the edge to a list.

Table 8.1: Key techniques used by Sherpa.

	TECHNIQUE	ORIGIN
3DC	**Three-dimensional clean-up**	FGR + **New**
BAS	Boxed A*-search	Well-known
BFR	**Branch-free representation for routed nets**	**New**
CNL	**Cyclical net locking**	**New**
CNR	Continual net restructuring	FastRoute [110]
DALS	**Dynamic adjustment of Lagrange steps**	**New**
ESH	ϵ-sharing	FGR
ELM	Edge-centric Lagrange multipliers	FGR
FLA	Fast layer assignment	FGR
LPR	**Logarithmic penalty function**	**New**
MST	Net decomposition by MST	Well-known

During the rip-up process for a subnet, each routing edge used by the subnet is examined (in an arbitrary order). For each such edge, first, the subnet is removed from the list maintained by the routing edge. Next, the map maintained by the parent of the subnet (the net to which the subnet belongs) is adjusted to reflect that one of its subnets no longer uses the edge. If no other subnets of the parent use the edge, it is removed from the mapping and resources are returned to the edge. Lastly, the routing edge is removed from the list maintained by the subnet. When adding a new route to a subnet, a similar sequence of steps is performed *in reverse*.

8.2 Analysis of Single-net Routing Techniques

In this section we analyze previously published techniques for single-net routing so as to select for Sherpa those methods which are both powerful and admit straightforward implementations.

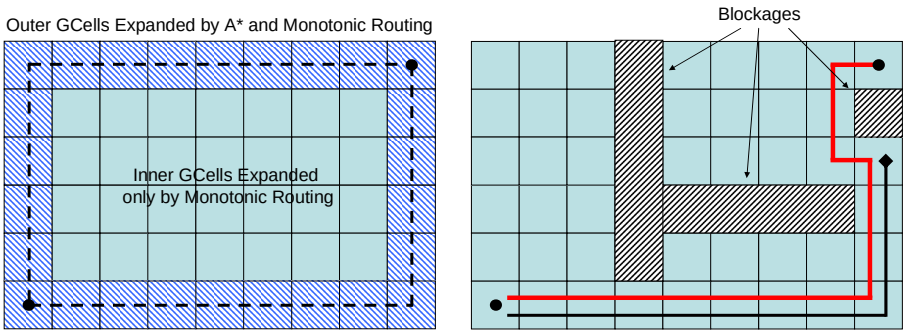


Figure 8.3: Boxed A*-search versus monotonic and pattern routing. On the left, we show an instance of the shortest- path problem with high bend costs. Boxed A*-search with a Manhattan lower bound searches fewer grid cells than monotonic routing to find the same solution. On the right, blockages obstruct the path and cause monotonic routing to fail, but boxed A*-search succeeds.

8.2.1 Point-to-point Maze Routing

Several options are available in the published literature for connecting pin pairs along the routing grid. Common methods include pattern routing (used by BoxRouter 1.0 & 2.0 [42] and Archer [108]) and monotonic maze routing (used by FastRoute 2.0 [110] and Archer [108]). In pattern routing, only certain route shapes are examined to connect points on the routing grid. Typically these shapes are chosen to have shortest wirelength and few bends such as “L” and “Z” patterns. Archer also uses slightly detoured “U” patterns [108].

In monotonic maze routing, the search space of maze routing is limited to the bounding box of the pins being routed. As an added restriction, only those edges that move closer to the target in terms of Manhattan distance are traversed. This can greatly speed up maze routing, but has several drawbacks. Some of these drawbacks are illustrated in Figure 8.3 where we compare monotonic maze routing with boxed A*-search (BAS). A*-search combines Dijkstra’s shortest path algorithm with a lower bound function to improve search

speed.[2] During BAS, search is also restricted to the bounding box of the pins, but all edges are allowed to be traversed. As Figure 8.3(b) illustrates, routing blockages can cause monotonic routing to terminate without finding a solution, whereas BAS finds a path with minimal detouring. Indeed, routing solutions found by monotonic routing are a subset of those found by BAS.

Even though boxed A*-search has a higher asymptotic complexity than monotonic routing (adding a logarithmic term due to the use of priority queues implemented with binary heaps), BAS with an admissible function can be faster than monotonic routing, as shown in Figure 8.3(a). When minimizing total wirelength and bends without blockages, BAS with a Manhattan-distance lower bound searches fewer grid cells to find the same solution as monotonic routing. Comparing monotonic routing to BAS, for most nets BAS appears almost as fast, is never inferior in solution quality and covers more cases, especially when congestion and blockages make monotonic routing undesirable. Compared to pattern routing, BAS can be much slower on very large nets, and we address this problem by developing a novel Cyclical Net-Locking (CNL) technique in Section 8.3.5 below.

8.2.2 Net Splitting

Most competitive routers decompose nets using Steiner tree construction algorithms. Given our use of the branch-free representation (BFR) for routed nets, it is natural for Sherpa to initially decompose nets into MSTs. Decomposition into MSTs can make routing more difficult as MSTs can have up to 150% of the wirelength of Steiner trees. Thus the maze router must work harder initially to reduce wirelength while also combating

[2] Bends are modeled as graph edges and priced independently.

congestion. To produce high quality routes with low wirelength, we restructure nets as described in Section 8.2.3 below. According to experiments in Chapter VII, initial decomposition by MSTs is a highly competitive strategy in terms of runtime and interconnect optimization. It increases flexibility in routing by avoiding the numerous flat nets present in Steiner minimal trees. Additionally, it facilitates a stand-alone implementation without relying on external Steiner-tree packages.

8.2.3 Continual Net Restructuring

Published competitive routers (Archer [108], BoxRouter 2.0 [42], FastRoute [110] and FGR) employ net restructuring, but in vastly different ways. Archer uses a sophisticated algorithm, based on net-length limits and Lagrange relaxation, that is restricted to the Hanan grid. This technique is expensive in runtime and therefore applied only once every $k = 50$ iterations. In contrast, FastRoute and FGR restructure nets continually during maze routing. We found that restrictions to the Hanan grid undermine interconnect optimization and unnecessarily decrease routing options. Therefore, we do not impose any restrictions on routes, restructure nets continually similar to FastRoute and FGR, while using the ϵ-sharing technique from FGR given its synergy with BFR. We recognize that similar design decisions were apparently responsible for somewhat higher runtimes in FGR results than in Archer results, and therefore address this problem in a fundamentally new way using our Cyclical Net-Locking (CNL) technique described in Section 8.3.5 below.

8.2.4 Handling Multi-layer Routing

When routing on a grid with multiple layers, there are two basic approaches. The first approach is to employ maze routing on the entire three-dimensional routing grid. The

second approach starts by projecting the three-dimensional routing grid onto a simpler two-dimensional grid and aggregating routing resources. After the two-dimensional routing grid is built and routing edges have proper edge capacities assigned, maze routing is performed. When maze routing completes, three-dimensional routing solutions for each net are reconstructed from solutions obtained on the two-dimensional grid.

In Section 7.1.3 we proved that if edge capacities are aggregated properly, a three-dimensional solution exists that has the same number of violations as the two-dimensional solution, and present a fast and greedy layer assignment algorithm that finds solutions for each net individually. This step is followed by a round of full three-dimensional clean-up when solutions are completely legal. In contrast, BoxRouter 2.0 builds a sophisticated ILP instance to solve the same problem. Given the simplicity of FGR's solution and relative solution quality difference between FGR and BoxRouter 2.0, we chose to implement FGR's solution. We supplement this technique with improvements to FGR's clean-up pass as described in Section 8.3.6 below.

8.3 Key Algorithms in Sherpa

Here we outline core algorithms used by Sherpa and highlight several aspects that we found critical to achieving improved runtime and solution quality.

8.3.1 The Sherpa Flow

A major challenge in large-scale routing is balancing wirelength against violations as competing objective functions. To this end, published routers include separate modules to balance wirelength and congestion [101, Section 3.4] by tuning weights in linear combi-

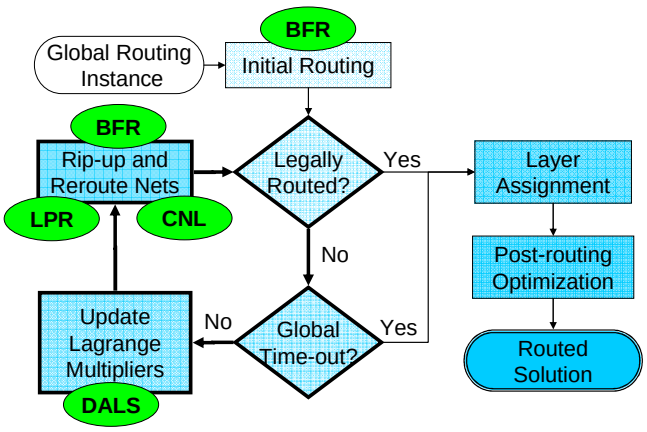


Figure 8.4: Global routing in Sherpa and the use of novel techniques such as a branch-free representation (BFR) for routed nets, cyclical net locking (CNL), dynamic adjustment of Lagrange steps (DALS) and a logarithmic penalty function (LPR).

nations. However, as articulated in [42], *ad hoc* trade-offs may lead to violent divergence of routing iterations. Therefore, several routers use *dampening* factors to ensure convergence [42, 108], intuitively similar to the cooling of temperature in simulated annealing.

The approach used in Sherpa is quite different in that the entire framework is structured around balancing wirelength and violations. This iterative framework, depicted in Figure 8.4, is based on Lagrange multipliers and achieves such an accurate equilibrium in practice that no dampening factors or separate guardian modules are required.

The key technique we use is *edge-centric Lagrange multipliers*, introduced first in FGR. While Lagrangian relaxation has been suggested for global routing before, all uses we are aware of are either (1) specific to timing-driven routing and maintain net-centric Lagrange multipliers [90] or (2) focus on a single net at a time [108]. These algorithms use conventional history-based rip-up and reroute for the router's main loop. In contrast, the Lagrangian formulation we use directly handles the global routing problem from the ISPD 2007 contest. In our formulation, the cost of a routing edge e is a function of a base

cost for the edge b_e, a Lagrange multiplier h_e and a penalty for local congestion p_e

$$c_e = b_e + h_e \cdot p_e \tag{8.1}$$

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

$$h_e^k = \begin{cases} h_e^{k-1} + h_{step} & \text{if } e \text{ is overfull} \\ h_e^{k-1} & \text{otherwise} \end{cases} \tag{8.2}$$

Our techniques differ from FGR in that we use a very different penalty function p_e for local congestion, as described in Section 8.3.4, and we do not use a constant h_{step}, which is examined in Section 8.3.3 below. Additionally, Equation 8.1 is different from that used by NCR in PathFinder [98].

The stopping criterion for rip-up and reroute iterations gauges the amount of effort applied on hard-to-route instances. To this end, the default version of Sherpa stops when a legal solution is found, after 50 rip-up and reroute iterations show no improvement, or upon running for 24 hours.

8.3.2 A Dual Lagrange Formulation

One flaw in both the NCR and DLM routing flows, presented in Chapters III and VII, respectively, is the following: once a net has become successfully routed through uncongested regions of the routing grid, it will not be re-examined for the remainder of the RRR iterations. After a legal solution has been found, we add a single round of greedy optimization described in Section 7.1.5 for each net, but this optimization could be too late. In other words, DLM and NCR target only those nets which pass through highly congested regions and not nets which may be detoured more than necessary.

To be certain that DLM pays more attention to those nets which are detoured, we must modify the routing formulation. First we reproduce our initial routing formulation, Equations 7.1, 7.3, and 7.4:

$$\min_{\mathbf{x}\in X} W(\mathbf{x})$$

$$\text{subject to } C_e(\mathbf{x}) = 0, \;\; 1 \leq e \leq n$$

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

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

where $C_e(\mathbf{x})$ is the amount of overflow on edge e, $W(\mathbf{x})$ represents the total wirelength of routing solution $\mathbf{x}$, $R_i(\mathbf{x})$ is the number of segments used by net i and $B_e(\mathbf{x})$ is the number of nets passing through edge e and b_e is the base cost of a routing edge. To focus the attention of DLM more on detoured nets, we can set up soft length constraints per net[3]

$$R_i(\mathbf{x}) \leq \alpha_i s_i \tag{8.3}$$

where s_i is the optimal Steiner length of net i and α_i is a multiplier that is set to a reasonable value such as 1.5 or can be user specified. These additional soft constraints call for additional Lagrange multipliers in the relaxation of the problem

$$F(\mathbf{x}, \lambda, \Lambda) = \sum_{e=1}^{n} \left(B_e(\mathbf{x}) + \lambda_e C_e(\mathbf{x}) \right) + \sum_{i=1}^{m} \Lambda_i L_i(\mathbf{x}) \tag{8.4}$$

where $L_i(\mathbf{x})$ represents by how much net i violates its soft length constraint.

This introduces a new weight for each net (illustrated in Figure 8.5), much like user-defined weights which were introduced in Section 7.1.1. We propose to treat the new Λ

[3]We call them soft constraints as we will not be strictly enforcing them, which is explained later in this section.

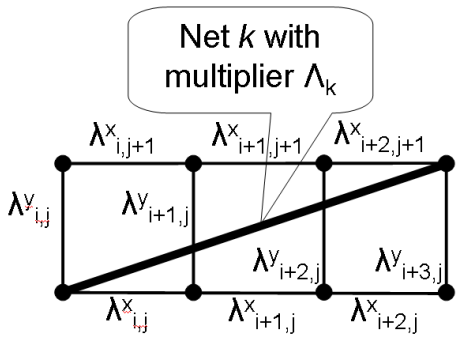


Figure 8.5: Relevant multipliers in the dual Lagrange formulation. Like the original formulation, each edge of the routing grid has a multiplier λ. In the dual formulation, each net also has its own multiplier Λ.

multipliers in the same way as net weights.[4] The rip-up-and-re-route iterations of DLM are modified in the following way: when historical congestion costs are updated for each routing edge, nets which violate their soft length constraints will have their net weights increased. Also, nets which violate their soft constraints will be ripped-up during RRR iterations in the same way that congested nets are ripped-up. We do not change the stopping criteria for all of the RRR iterations. As the soft length constraints that were added may not be strictly realizable depending on the choice of α, it is acceptable if they are violated as long as all edge capacity constraints are satisfied.

8.3.3 High-precision Lagrange Multipliers

Lagrange multipliers are critical to the success of the NCR [98] and DLM (Section 7.1.1) routing frameworks and are a dominant factor in determining solution quality as well as routing runtime. Thus it is critical to precisely determine Lagrange multipliers during rip-up and reroute to achieve a good runtime and solution quality trade-off.

Previous work [98] increases Lagrange multipliers of congested edges by a constant

[4] If a non-trivial weight is specified for a net, the total weight for that net will be the product of the Lagrange weight and the user-supplied weight.

h_{step} according to Equation 8.2. In our experiments, we find that, in general, 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.

To find better Lagrange steps for arbitrary benchmarks, 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 8.3.1 and Figure 8.4, while Lagrange steps are modified between iterations as follows

$$h_{step}^{k+1} = \begin{cases} h_{step}^{k} + \Delta_{step} & \text{if } viol_k \geq viol_{k-1} \\ h_{step}^{k} - \Delta_{step} & \text{if } viol_k < viol_{k-1} \text{ and } WL_k > WL_{k-1} \\ h_{step}^{k} & \text{if } viol_k < viol_{k-1} \text{ and } WL_k \leq WL_{k-1} \end{cases} \tag{8.5}$$

Empirically, Lagrange steps change significantly during the early iterations of rip-up and reroute, settle to within a small range of steps during the middle iterations, and finally increase when nearing a legal solution. As reported in Table 8.2, DALS preserves the solution quality of FGR while contributing to significant runtime speedups and reduced violation counts.

8.3.4 Logarithmic Penalty Function

The penalty function used by FGR grows linearly while an edge is legal with respect to capacity constraints and grows *exponentially* with relative overflow afterward. Relative overflow is defined as the fraction of routing resources used by an edge. For example, a completely unused edge has a relative overflow of 0, while an edge with 50% more usage

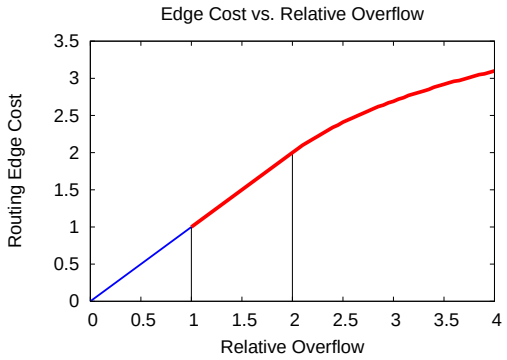


Figure 8.6: New *convex* penalty function used by Sherpa. The function grows linearly until a routing edge uses 200% of its routing resources and logarithmically thereafter. This radical departure from *concave* penalty functions used by other routers is made possible by the strength of the underlying global routing algorithm and improves the handling of designs with numerous violations.

than capacity has a relative overflow of 1.5. This function is effective at reducing total and maximum overflow when solutions are nearly legal, but it hampers the performance of A*-search when a solution is highly illegal, e.g., for the `newblue3` benchmark. We have found the penalty function effectively nullifies the Manhattan lower bound given to A*. This makes A*-search much slower, resulting in very few iterations of rip-up and reroute during the 24-hour timeout period.

To counteract this problem, we introduce a new penalty function p of a routing edge e based on the relative overflow ω of e

$$p(e) = \begin{cases} \ln(\omega - 1) + 2 & \text{if } \omega > 2 \\ \omega & \text{otherwise} \end{cases} \tag{8.6}$$

Our penalty function grows linearly until a routing edge uses 200% of its routing resources and then logarithmically, i.e., more slowly, thereafter, which is shown in Figure 8.6. This function is fundamentally different from others presented in the literature in that it is *convex*.[5] In contrast, *concave* penalty functions, especially exponential functions, emphasize

[5]Since a convex penalty function puts less pressure on maximum violations, it may not work well with a

the minimization of maximum overflow rather than total overflow. We believe this is a key reason that FGR produces poor violation counts on the `newblue3` benchmark, which can only be routed with a large number of violations. Use of a *convex* penalty function leads to dramatically reduced violations on difficult instances compared to FGR.

8.3.5 Cyclical Net-locking

We observed through profiling that the vast majority of runtime in the unmodified Sherpa flow is spent routing nets with long wirelength. The ISPD 2007 netlists follow the standard Rentian wirelength statistics, whereas the number of nets with given length decreases geometrically with length. Therefore, we refocus the framework of Section 8.3.1 to short nets and reroute long nets less frequently.

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

$$\begin{aligned} Area(\text{BBox}_n) &= (|\text{BBox}_n.x1 - \text{BBox}_n.x2| + 1)\times \\ &\quad (|\text{BBox}_n.y1 - \text{BBox}_n.y2| + 1) \end{aligned} \tag{8.7}$$

This effectively estimates the search space for boxed A*-search on a subnet, and draws upon our observations that (1) almost all nets route within 2x of their HPWL and (2) very few nets route with significant detours.

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 in comparison with the average bounding box size:

$$AvgArea = \left(\frac{1}{N}\right) \sum_{n=1}^{N} Area(\text{BBox}_n) \tag{8.8}$$

weaker baseline router.

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

$$Period(n) = \min\left\{\left\lceil \frac{Area(\text{BBox}_n)}{AvgArea} \right\rceil, 10\right\} \tag{8.9}$$

Thus large subnets are unlocked less frequently than small subnets (but at least every 10 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

$$(i < 2) \text{ or } ((i + n) \bmod Period(n) = 0) \tag{8.10}$$

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 8.3.1 dramatically with very little impact on solution quality.[6] The success of CNL shows there is significant flexibility in choosing which nets to reroute to avoid congested regions, and that focusing on shorter nets is more efficient.

8.3.6 Multi-layer Routing

The three-dimensional clean-up pass after layer assignment described in Chapter VII is only applicable to legally routed instances; if a net had an illegal solution at the end of the rip-up and reroute phase, there is no guarantee that a solution is possible that only uses uncongested routing edges during the clean-up stage. We note that restricting nets to pass through only legal routing edges is overly limiting and modify the technique to improve both legal and illegal solutions.

[6] 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.

After layer assignment and before traditional three-dimensional 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 as normal and find a solution that uses less wirelength. Note that the total and maximum overflow values of the original solution cannot increase since no illegal routing edge is allowed to increase in illegality and no legal routing edge can become illegal. After clean-up, we reinstate the correct capacities for all routing edges and recalculate the total and maximum overflow statistics for the solution. We observe that this clean-up method is as effective in reducing total wirelength usage in illegal solutions as it is in legal solutions, and usually decreases total overflow by a small amount as well.

8.4 Empirical Evaluation

We implemented Sherpa in C++, and used g++ 4.2.2 to produce 64-bit binaries for our experiments.[7] All experiments were run on 2.4GHz AMD Opterons with 4GB of RAM.

We point out that many publications in placement and routing report results produced by tuning software to individual benchmarks. For example, results of Archer were tuned to reduce runtime [108, Section 6] on the `newblue3` benchmarks, even when presenting results of a "default" configuration [108, Table 1]. In other publications, such tuning is not stated explicitly. Our Sherpa router is able to achieve very strong results in default configuration.

Table 8.2 compares Sherpa with Archer [108], BoxRouter 2.0 [42] and FGR on the ISPD 2007 Global Routing Contest benchmarks [71]. Where possible, we compare against

[7] We confirmed with our main competitors, Archer and FGR, that they also used g++.

Table 8.2: Sherpa compared with published results of Archer [108], BoxRouter 2.0 [42] and FGR on the ISPD'07 benchmark suite [71]. Sherpa is run in a default configuration for each benchmark. For Archer and FGR, we compare against un-tuned results where runtimes were reported. (†) Results of Archer were produced on 3.6GHz Intel Xeon processors while FGR and Sherpa were run on 2.4GHz AMD Opterons. We ran speed tests on similar machines and them to be 1.67x faster, so Archer runtimes have been scaled up by 1.67x to facilitate comparisons. (‡) According to [108, Section 6], Archer did not use a default configuration on `newblue3`, despite the claim in the caption of [108, Table 1]. Thus, we do not include `newblue3` in runtime comparisons with Archer.

Benchmark	Archer [108]			BoxRouter 2.0 [42]			FGR				Sherpa			
	Overflow total	Cost (e5)	Time† (m)	Overflow total	Overflow max	Cost (e5)	Overflow total	Overflow max	Cost (e5)	Time (m)	Overflow total	Overflow max	Cost (e5)	Time (m)
adaptec1 2-d	0	58.27	143	0	0	58.37	0	0	**54.44**	451	0	0	54.86	**57**
adaptec1 3-d	0	113.80	145	0	0	92.04	0	0	**88.45**	430	0	0	88.64	**62**
adaptec2 2-d	0	54.60	37	0	0	55.69	0	0	**52.30**	56	0	0	52.36	**18**
adaptec2 3-d	0	112.56	38	0	0	94.28	0	0	89.89	64	0	0	**89.88**	**23**
adaptec3 2-d	0	135.42	82	0	0	137.96	0	0	**130.89**	179	0	0	131.26	**52**
adaptec3 3-d	0	244.08	85	0	0	207.41	0	0	**199.66**	243	0	0	199.71	**70**
adaptec4 2-d	0	126.26	18	0	0	127.79	0	0	125.00	19	0	0	**124.84**	**8**
adaptec4 3-d	0	221.57	20	0	0	186.42	0	0	**182.96**	55	0	0	183.07	**18**
adaptec5 2-d	0	162.49	410	0	0	162.11	0	0	**152.13**	713	0	0	152.93	**224**
adaptec5 3-d	0	334.09	413	0	0	270.41	0	0	**259.98**	740	0	0	260.24	**229**
newblue1 2-d	682	48.40	**82**	400	2	51.13	452	4	47.43	1441	**374**	4	**46.29**	323
newblue1 3-d	682	116.08	**83**	394	2	92.94	452	2	94.27	1442	**374**	2	**90.42**	348
newblue2 2-d	0	77.91	10	0	0	78.68	0	0	76.51	4	0	0	**76.37**	**2**
newblue2 3-d	0	166.50	12	0	0	134.64	0	0	**132.16**	10	0	0	132.19	**4**
newblue3 2-d	**33394**	109.25	(270)‡	38958	1088	111.61	38580	1120	109.34	1555	35044	1192	**107.24**	1457
newblue3 3-d	**33394**	198.77	(272)‡	38958	364	172.44	38580	374	173.82	1501	35042	398	**163.36**	1474
Average 2-d		+3.7%	1.71x			+5.4%			+0.4%	2.97x				
Average 3-d		+25.1%	1.31x			+3.6%			+1.3%	3.02x				
Overall Average		+13.9%	1.50x			+4.5%			+0.8%	3.00x				

default results of each of the other tools. Sherpa's solutions improve on FGR's solutions overall by 0.8% and are never more than 1% worse on any benchmark. Sherpa also produces the best published violation counts on the `newblue1` benchmarks. Overall, Sherpa outperforms BoxRouter 2.0 by 4.5% and Archer by 13.9% in wirelength. Note that Archer is very competitive on two-dimensional benchmarks, but on three-dimensional benchmarks it is 25% worse than Sherpa. This is due to our improved three-dimensional clean-up phase after layer assignment. FGR lags behind Sherpa significantly in runtime and in violation counts on the `newblue3` benchmarks. This is because of our CNL and LPF techniques, respectively.

While FGR and Sherpa runs were performed on 2.4GHz AMD Opteron machines, Archer was run on 3.6GHz Intel Xeon processors. We ran speed tests on similar Intel machines and found those machines to be 1.67x faster, so Archer runtimes have been scaled up by 1.67x to facilitate comparisons. Sherpa produces the fastest runtimes on 12 of the 16 benchmarks. Overall, Sherpa is 50% faster than Archer (not including `newblue3` as Archer runs were tuned for runtime and Sherpa ran for a 24-hour period). On `newblue1`, default Sherpa is not faster than Archer, but Archer finishes routing with nearly twice as many violations as Sherpa and 4% higher wirelength. Sherpa can produce the same violation counts as Archer in 72 minutes. Factoring in these `newblue1` runtimes, Sherpa is 2.2x faster than Archer on two-dimensional instances, 1.6x faster on three-dimensional instances, and 1.9x faster overall. Compared to FGR, Sherpa is 3.0x faster on two- and three-dimensional instances.

8.5 Conclusions

We have presented the Sherpa global router which outperforms ICCAD'07 results of Archer [108], BoxRouter 2.0 [42] and FGR in terms of runtime, while matching best wirelength results and reducing violation counts. Sherpa uses a Lagrange relaxation routing framework with several key enhancements including a branch-free representation (BFR) for routed nets, a logarithmic penalty function (LPF), dynamic adjustment of Lagrange steps (DALS) and a cyclical net-locking (CNL) technique.

PART IV

Placement and Routing in Modern Design Flows

CHAPTER IX

Integration of Routing into Placement and Physical Synthesis

Dramatic progress has been made in algorithms for placement and routing over the last 5 years, with improvements in both speed and quality. Combining placement and routing into a joint optimization has also been proposed. However, it remained unclear until now if the benefits would be significant enough to justify major changes in commercial tools. Our work addresses this challenge and is the first to demonstrate tangible benefits of combined place-and-route optimization including fewer global routing detours, reduced detailed routing violations and runtime, and even shrinking the floorplan of a commercial design. We employ fast global routing to choose standard cells to temporarily inflate and

iteratively spread for congestion reduction. Spreading cells only in congested regions, our technique CRISP (Congestion Reduction by Iterated Spreading during Placement) enables die area reduction by facilitating routing with high area utilization.

9.1 Industrial Physical Design

Physical design has been a major bottleneck in modern EDA flows, and its significance is increasing for large ICs due to the poor scaling of interconnect delay (relative to transistor delay). The focus of our work is on extremely large ASIC and SoC designs that contain multiple millions of standard cells and thousands of macro blocks. Critical path delay in such designs is often dominated by interconnect, and the choice of cell locations affects circuit delay to a large extent, followed by the routes chosen for the longest wires. These degrees of freedom correspond to *placement* and *routing*, which have traditionally been handled by independently-developed EDA tools. Due to the significance of these optimizations, they received a great amount of attention from researchers, and the contests organized by IBM at ISPD confirmed dramatic improvements in the speed and quality of placement and routing on large industry netlists, achieved by university researchers [105,106]. Not only have the basic algorithms changed in the last 5-10 years, but the very landscape of placement and routing has changed dramatically. Perhaps, the most visible difference from older ASICs is the presence of large amounts of *whitespace*, inserted to support net buffering, gate sizing, post-placement logic restructuring and ECOs, as well as to limit power density and to ease routing. Another prominent feature of modern ASICs and SoCs is the presence of large fixed macros and routing obstacles.

Design objectives. Most work on congestion mitigation in placement cites as its primary motivation the need to facilitate violation-free routing (global or global+detailed) and decrease turn-around time (TAT) by reducing design iterations. Publications also point out that interconnect length must not increase too much while routability improves [9, Chapter 22]. While this trade-off is important to our work, we point out that additional motivating factors can be more critical to success in an industry environment. Indeed, total interconnect length (before or after routing) is rarely a goal in itself, but is rather viewed as a proxy for circuit delay and dynamic power. Given that a large fraction of dynamic power is due to clocks, and total dynamic power is decreasing compared to the contribution of static power, circuit delay is viewed as the key objective. In particular, timing-driven placement algorithms can be improved by early delay estimates, whose accuracy has a direct impact on the quality of circuit delay optimization. Such estimates use pre-routes for individual nets to fully account for the capacitance of Steiner tree and other effects. When congestion estimates are available, pre-routes can be constructed to avoid congested regions, improving the quality of delay estimation. However, constructing pre-routes independently per net often results in multiple pre-routes occupying the same track, especially near obstacles. Therefore pre-routes should be generated by a global router.

Chip area considerations. An important new consideration in our work is related to chip size. By working with entire ASICs and SoCs, rather than isolated partitions, one can control the area and the shape of the chip. In our case, chip floorplans are created by expert design engineers who account for many factors, including the whitespace required for routability, as well as the placement of fixed macro blocks. Experiments discussed in

Section 9.3 demonstrate a strong place-and-route tool that can handle high area utilization and thus requires less whitespace in the floorplan. To help us extract maximum benefits from such a tool, expert design engineers working with us produced alternative floorplans with smaller area, and we demonstrate that such floorplans can be satisfied in terms of violation-free routing and timing closure. When using 300mm wafers, decreasing chip area by 5% increases the number of chips in one wafer, decreasing the cost of each chip [9, Part VII].

Compatibility with existing EDA infrastructure. On the algorithmic side, we are working with a state-of-the-art timing-driven force-directed placement framework and enhance it by accounting for congestion and routability. Unlike in previous published work on combined placement and routing [45], we consider and evaluate these steps within a complete industrial physical-synthesis flow. This imposes a series of compatibility requirements, but also allows us to draw upon available netlist transforms. Our contributions include a new technique for interconnect estimation, the use of incremental cell inflation, and the use of dedicated legalization and detailed placement. The need for dedicated steps is due to the requirement to preserve timing and the validity of neighboring optimizations.

Key contributions of our work include:

- We present CRISP, an incremental placement technique which improves the routability of a given placement through highly accurate congestion modeling. CRISP takes advantage of this accuracy and constructs a novel measure-and-improve incremental placement flow.
- We formulate exact requirements for determining when a region has too many pins and propose a separate placement-spreading pass driven by pin density, applied be-

tween global and detailed routing. As intended, this pass improves detailed routing in our experiments while preserving global routes. This work is the first to separately measure and optimize global and detailed routability in placement.

- We apply CRISP to layouts produced by mPL6 [28] for the ISPD contest benchmarks [105, 106] and route them using NTHU-Route 2.0 [33]. CRISP improves via counts by 8.7%, global routed wirelength by 6.5% and detouring by 5.3%.
- We use CRISP in an industrial physical-synthesis flow to make previously unroutable designs routable, reducing detailed routing runtime, detours and violations.
- We illustrate that effective congestion reduction can be applied to shrink the die size of a commercial design by 5%. Thus, CRISP leads to savings in manufacturing cost.

The remainder of this chapter is organized as follows. Section 9.2 describes our proposed techniques known as Congestion Reduction by Iterated Spreading during Placement (CRISP). Section 9.3 validates CRISP in experiments on publicly available benchmarks with academic placers and routers as well a large commercial ICs. Section 9.4 summarizes our work and concludes.

9.2 CRISP Techniques

We seek an incremental technique which can be applied to any placement in a physical-design flow to reduce congestion as well as preserve timing characteristics. To this end, we present CRISP, a technique which reduces routing congestion by incremental placement changes based on highly accurate congestion metrics. CRISP assembles state-of-the-art placement and global routing techniques into a new incremental placement flow, adding

several missing pieces. CRISP carefully spreads standard cells located in congestion hotspots while preserving the placement of uncongested areas where possible. If spreading creates new congested areas, CRISP iterations identify and eliminate them. In this section, we describe the techniques used by CRISP to model routing congestion, spread the placement and dissolve congested spots.

9.2.1 Modeling Routing Congestion

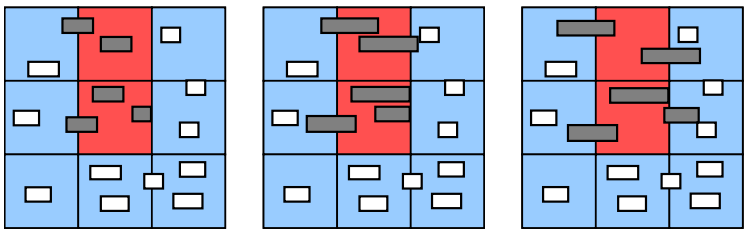
Using probabilistic congestion maps, while computationally efficient, suffers from two important drawbacks: (i) it does not account for routing blockages, and (ii) it does not allow for detouring. In the presence of numerous fixed routing blockages, common in modern designs, using a probabilistic congestion map can impair routability as we demonstrate in Section 9.3.

Rather than build a probabilistic congestion map, CRISP creates a global routing instance from the current placement and uses a global router to generate a full set of routes. Prior to the ISPD 2007, 2008 routing contests [105, 106], most publications assumed that global routing was too expensive to invoke during placement. However, we demonstrate that recent advances in global routing algorithms make them affordable as estimators. To keep routing runtime practical, CRISP limits the amount of detouring the global router is allowed to perform. This allows CRISP to capture the areas of the design which have actual rather than estimated routing congestion as well as identify areas of congestion which can be caused by detouring. An example of a congestion map derived from an academic global router is shown in Figure 9.4(b), and a congestion map from an industry global router is shown in Figure 9.5.

Routability metrics. An important consideration when modeling congestion is being able to determine which of two placements is more routable. At the ISPD global routing contests, the amount of *overflow* in a routing solution was the primary quality metric. Overflow is defined as the difference between routing edge usage and capacity for all routing edges that use more than their capacity. In industrial tools, overflow is discarded in favor of net-based metrics. For each net, one can derive the maximum congestion of any routing edge used by the net. Using this *net congestion* metric, one can distinguish nets whose congestion exceeds a given threshold. For example, the number of nets which are at least 100% congested refers to all nets passing through at least one routing edge which uses 100% or more of its routing resources.

Accounting for pin density. Local peaks of pin density often cause routing congestion, but are overlooked as a source of congestion by many algorithms. Global routing accurately captures the wires that pass between routing edges, but does not focus on congestion internal to GCells. In fact, this source of congestion is ignored completely by academic global routing formulations, and, in our experience is underestimated by industry global routers. A design may appear to be easily globally routable, but may fail detailed routing leaving several shorts and opens. We propose to handle pin density directly in CRISP with a separate pass of placement-spreading driven by pin density applied before detailed routing.

A naive method to mitigate the inaccuracy of global routing with respect to pin density is to shrink GCell sizes so that fewer nets become subsumed by GCells, but this can make global routing too slow for practical use. Our solution assimilates and extends ideas from



(a) (b) (c)

Figure 9.1: (a) A placement with two congested areas. (b) CRISP inflates standard cells in these regions, (c) and spreads them.

previous work [16, 124], which injects whitespace into areas of high pin density or routing congestion. Novel elements include (i) exact requirements for determining when a region has too many pins and (ii) a separate placement-spreading pass driven by pin density, applied before detailed routing. The latter seeks to ease detailed routing while preserving global routes.

9.2.2 Temporary Cell Inflation

The technique of cell inflation is used by experienced designers to alleviate routing congestion, and it proves to be very effective in practice [16, 124]. For this reason, we develop algorithms for cell inflation in the context of incremental placement. During each iteration of CRISP, we determine areas of congestion and inflate cells in the most congested areas preferentially. Thus cells which are consistently found to be in heavily congested regions grow in size more quickly over time than those in light congestion, which is illustrated in Figure 9.4(c).

We inflate cells in proportion to their pin counts in order to reduce pin density in congested regions. Empirically this step improves detailed routability (see Section 9.3).

The width of cell c during iteration i, width(c, i), is

$$\max(\text{width}(c, i-1) + 1, \lceil (1 + \alpha T)\text{numPins}(c) \rceil)$$

where T is the number of times c has been in a congested region, α is the width increment and width$(c, 0)$ is the initial width of c. An example of CRISP inflating cells in shown in Figure 9.1. We use the same $\alpha = 0.2$ for all of our experiments in Section 9.3.

9.2.3 Incremental Spreading

Spreading techniques are common in the literature for analytical placement algorithms. One such technique is iterative local refinement (ILR) used by FastPlace [137]. ILR creates a regular grid for a given placement and performs many rounds of movement for every cell in a design. During each round, each movable cell is examined once. A cell may move from its current grid tile to one of its eight neighboring grid tiles. The choice of destination for each cell is based on a cost function which is a linear combination of the change in wirelength and area balance between the source and destination tiles caused by the move.

The FastPlace iterative local refinement framework is insufficient for use in CRISP because any cell can be moved at any iteration. We enhance ILR for use in CRISP and remove this limitation. For each grid tile, we define a target density and a multiplier describing the relative importance of area requirements versus wirelength for the tile. At the beginning of each round, tiles which are above their target density have their multiplier increased so that satisfying their density constraint becomes more important than change in wirelength; tiles which meet their density constraint have their multipliers reduced. During each round, only movable cells contained within tiles that do not meet their density constraint are examined. Additionally, we impose a greedy ordering on cells so that those

with better gain in cost function are moved preferentially. When we call this modified ILR during CRISP (Figure 9.3, line 20), we assign the target density for a grid tile to be the density the tile had at the beginning of the iteration of CRISP before cells were inflated. Overall, these modifications ensure that areas with no congestion, and thus no cell inflation, will remain undisturbed during spreading, making ILR suitable for incremental placement.

Legalization and detailed placement. The fidelity of layout modeling is essential to the success of CRISP. If routing or pin-density hot-spots identified by the router do not correspond to actual areas of routing congestion, CRISP could do harm to the placement rather than help. Since legalization often changes the routabililty of a design, it is particularly import that all placements CRISP evaluates using a router be legal. Thus any solution returned by CRISP will necessarily be legal and any observed improvements in routability will carry over into the final result. Unfortunately, legalization in the presence of many fixed obstacles often significantly perturbs locations, increasing wirelength. To recover from wirelength gained during legalization we run detailed placement techniques after legalization. To save runtime as well as preserve the spreading of the placement, we limit the detailed placer to one round of its transforms.

9.2.4 The CRISP Flow

In Figure 9.2, we outline the flow of CRISP. Pseudocode is also given in Figure 9.3. An example of CRISP reducing congestion on a highly-congested commercial design is shown in Figure 9.5.

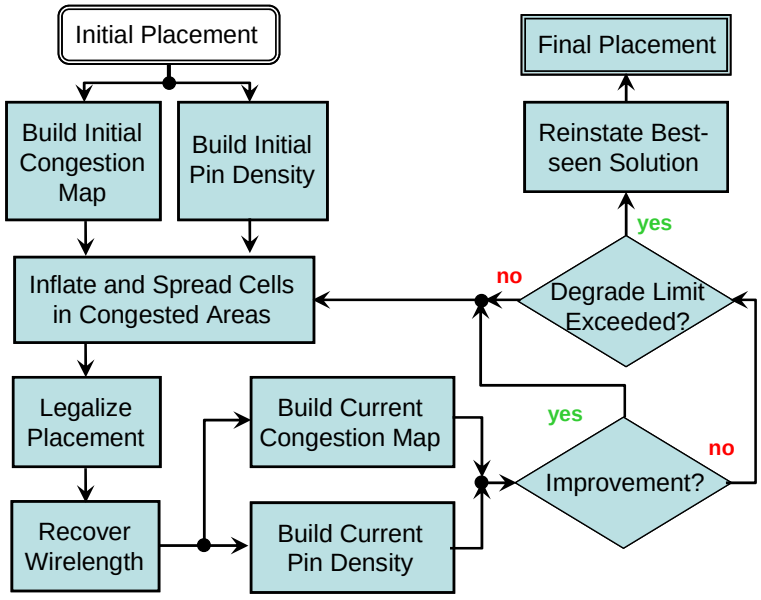


Figure 9.2: The CRISP incremental placement flow.

Congestion measurement. CRISP first determines the routing congestion of an initial placement by calling a fast and effective global router. To limit runtime, CRISP restricts the detouring that the router is allowed to perform. In the context of academic routing tools, CRISP limits the number of iterations of rip-up and reroute. When using an industrial router, CRISP can use more accurate constraints. Thus it limits the industrial router to 5% detouring (see Section 7.2). After the global router produces a solution, CRISP generates a congestion map and a pin-density map for the current placement. Using these maps, CRISP determines portions of the design that are problematic in terms of routing congestion or pin density. For pin density, the threshold for our experiments is chosen as 1 pin per minimum area of a standard cell (one standard row high and one site width wide). For routing congestion, we use different thresholds when using academic and industrial design tools. Since academic routers seek to reduce total routing overflow, we set the threshold to 95% congestion. For commercial designs, we wish to reduce the number of nets which

CRISP: CONGESTION REDUCTION BY ITERATED SPREADING DURING PLACEMENT

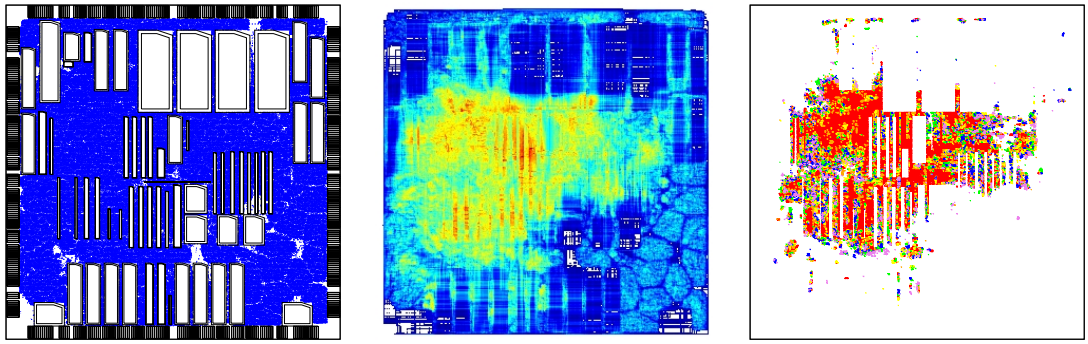
▷ Input: Placement P, Global router GR, Cell width increment α,
▷ Congestion target $congTarget$, Maximum iterations $maxIter$,
▷ Maximum area increase per iteration $maxAreaIter$
▷ Output: Congestion optimized placement $PBest$

create two arrays, *timesCongested* and *cellCong*,
having size numMovableCells(P), entries initialized to 0
if(isPlacementLegal(P) == FALSE)
then legalizePlacement(P), doDetailedPlacement(P)
congMap = callRouter(GR,P), *pinMap* = buildPinDensityMap(P)
$PBest = P$, *CongBest* = getCongMetric(*congMap*,*pinMap*)
currArea = getUsedCoreArea(), *totalArea* = getTotalCoreArea()
iters = *losingStreak* = 0, *originalWidth* = getCellWidths(P)
do
  foreach cell c
    cellCong[c] = max(lookUpCong(congMap,getLocation(P,c)),
lookUpCong(pinMap,getLocation(P,c))
  maxAreaThisIter = min(0.95·*totalArea*, *currArea* + *maxAreaIter*·*totalArea*)
  foreach cell c in order of decreasing congestion
    if(*cellCong*(c) < *congTarget*) **then break**
    timesCongested[c]++
    newWidth = max(getWidth(P,c) + 1, ⌈(1 + α·*timesCongested*[c])·getNumPins(c)⌉)
    if(*currArea* + (*newWidth* - getWidth(P,c))·getCoreRowHeight() >
      maxAreaThisIter) **then continue**
    currArea += (*newWidth* - getWidth(P,c))·getCoreRowHeight()
    setWidth(P,c,*newWidth*)
  spreadPlacement(P), legalizePlacement(P), doDetailedPlacement(P)
  congMap = callRouter(GR,P), *pinMap* = buildPinDensityMap(P)
  CurrCong = getCongMetric(*congMap*,*pinMap*)
  if(*CurrCong* < *CongBest*)
  then *CongBest* = *CurrCong*, $PBest = P$, *losingStreak* = 0
  else *losingStreak*++
  iters++
while(*iters* < *maxIters* **and** *losingStreak* < 2)
return $PBest$

Figure 9.3: The CRISP algorithm for determining which cells to inflate per iteration.

have 90% or more congestion and so set a threshold of 85% congestion.

Cell inflation and spreading. Next CRISP assigns a congestion number to each standard cell based on the routing and pin density in the regions it occupies. CRISP limits the amount of inflation that may happen during a particular iteration by imposing a user defined maximum area increase. Each cell is examined for inflation in order of decreasing congestion until all cells have been examined or the maximum area is reached for that iter-



(a) (b) (c)

Figure 9.4: (a) Placement of `adaptec1` with 60% target density and (b) corresponding congestion map. (c) Map of cells inflated during the first five iterations of congestion elimination. Colors in (c) correspond to relative inflation with red cells being the greatest followed by orange, yellow, green, blue and violet.

ation. For all of our experiments, we limit inflation to 1% of the core area per iteration. In addition, we impose a limit of 95% core area usage to ensure that a legal placement is feasible. After as many congested cells as possible are inflated under the area limits, CRISP spreads the newly inflated cells to make the placement more legal. The goal of spreading is to produce a nearly-legal placement by perturbing the original solution as little as possible, without significantly degrading wirelength. CRISP's inflation and spreading steps are illustrated in Figure 9.1, and described in Sections 9.2.2 and 9.2.3. After the solution has been spread, CRISP calls a legalizer followed by detailed placement to recover wirelength. Figure 9.4(b) shows a congestion map for an academic design and Figure 9.4(c) shows which cells are inflated and to what degree over several rounds of CRISP.

Iterative congestion reduction. CRISP then calls the global router and compares routability metrics with previous placements. If the new placement improves routability metrics, it is saved. CRISP iterations continue in this way until a stopping criterion is met. Stopping criteria include (i) a maximum number of iterations, (ii) a maximum number of

iterations in a row without congestion metric improvement, (iii) area restrictions preventing cell inflation, or (iv) complete elimination of congestion. CRISP returns the best-seen placement in terms of congestion metrics.

9.3 Experimental Results

We test CRISP and compare it with state-of-the-art congestion reduction techniques on a wide range of designs both academic and commercial. For academic designs, we choose the ISPD placement and routing contest benchmarks. We place and route these designs with academic tools and compare CRISP with academic incremental congestion reduction techniques. On commercial designs, we demonstrate how CRISP reduces global congestion and detouring, and improves detailed routability. We also show how CRISP can be used to shrink the die of a commercial design.

9.3.1 ISPD Contest Benchmarks

Benchmark setup. To test the effectiveness of CRISP, we placed the ISPD placement and routing contest benchmarks [105, 106] with the academic placer mPL6 [28]. mPL6 is an analytical placer which finished 2nd place overall at the ISPD 2006 placement contest. The 2006 placement contest featured density constraints for all benchmarks and mPL6 achieved the best total wirelength, while largely observing density constraints, but lost to the contest winner by runtime. For each benchmark, we produce two placements, one routable and one unroutable, by varying the density constraint passed to mPL6. The ISPD contest benchmarks range in size from 211,447 objects (543 fixed) for `adaptec1` to 2,507,953 objects (26,582 fixed) for `newblue7`. We exclude `newblue3` from our

Table 9.1: Using ECO-system [117], the Bonn flow [16] and CRISP to improve the routability of unroutable mPL6 [28] placements of ISPD contest benchmarks [105, 106]. We were unable to produce unroutable placements of bb3 or nb2 with mPL6. We exclude nb3 because it is trivially unroutable. Detouring is measured as the ratio of global routing segments to FLUTE [44] Steiner wirelength. † NTHU-Route crashed on three instances, ending CRISP prematurely.

Benchmark & Target density	Placement			Global routing (NTHU-Route 2.0 [33])						Reduction of		
	Flow	Runtime (min)	HPWL (e6)	Estimated overflow	Runtime (min)	Final overflow	Vias (e6)	RWL (e6)	Detour ratio	Vias	RWL	Detours
ad1, 80%	mPL6	61	81.7	174626	1030	560	1.98	5.12	1.220	—	—	—
	mPL6 + ECO-system	172	234.8	Placement illegal						—	—	—
	mPL6 + Bonn	126	83.0	152802	52	16	1.88	4.77	1.103	5.1%	7.0%	11.7%
	mPL6 + CRISP	88	83.1	148486	65	36	1.83	4.72	1.100	7.4%	7.9%	12.0%
ad2, 70%	mPL6	80	97.3	97224	>1440	8086	2.00	5.42	1.104	—	—	—
	mPL6 + ECO-system	307	323.7	6316606	>1440	4903640	3.05	17.14	1.427	-52.5%	-216%	-32.3%
	mPL6 + Bonn	159	100.0	70182	5	6	1.96	5.36	1.037	2.2%	3.0%	6.7%
	mPL6 + CRISP	108	96.1	66166	62	92	1.84	5.05	1.049	7.7%	6.7%	5.5%
ad3, 80%	mPL6	379	216.9	269594	44	38	3.90	12.16	1.049	—	—	—
	mPL6 + Bonn	695	219.7	227630	11	0	3.88	12.20	1.042	0.5%	-0.3%	0.7%
	mPL6 + CRISP	442	218.7	182836	9	0	3.70	11.97	1.038	7.5%	3.6%	1.1%
ad4, 90%	mPL6	221	191.2	97972	>1440	1266	3.62	10.98	1.078	—	—	—
	mPL6 + Bonn	460	192.1	84052	17	68	3.53	10.52	1.019	2.6%	4.2%	5.9%
	mPL6 + CRISP	267	192.3	49334	3	0	3.31	10.33	1.014	8.6%	5.9%	6.4%
ad5, 70%	mPL6	408	365.2	190026	56	4	5.67	13.91	1.036	—	—	—
	mPL6 + Bonn	785	372.0	150008	13	0	5.65	13.94	1.024	0.4%	-0.2%	1.2%
	mPL6 + CRISP	522	359.1	108950	6	0	5.19	13.17	1.020	8.5%	5.3%	1.6%
bb1, 60%	mPL6	74	113.9	169072	>1440	13296	2.27	5.87	1.372	—	—	—
	mPL6 + Bonn	151	116.2	125522	13	2	1.94	4.75	1.093	14.3%	19.0%	27.9%
	mPL6 + CRISP	109	114.9	116626	10	0	1.83	4.58	1.081	19.2%	22.0%	29.1%
bb2, 50%	mPL6	213	172.6	58908	7	4	4.05	9.06	1.026	—	—	—
	mPL6 + Bonn	402	175.2	49176	4	0	4.06	9.13	1.023	-0.2%	-0.8%	0.3%
	mPL6 + CRISP†	325	171.4	21776	2	0	3.73	8.70	1.019	8.0%	3.9%	0.7%
bb4, 50%	mPL6	728	954.7	49294	>1440	2102	13.47	27.04	1.026	—	—	—
	mPL6 + Bonn	1386	959.0	41250	48	212	13.46	26.90	1.010	0.0%	0.5%	1.6%
	mPL6 + CRISP†	959	929.7	12892	4	0	12.19	25.13	1.007	9.5%	7.0%	1.9%
nb1, 70%	mPL6	99	699.6	16990	7	80	2.07	4.69	1.016	—	—	—
	mPL6 + Bonn	190	699.4	11478	6	54	2.05	4.67	1.012	0.7%	0.4%	0.4%
	mPL6 + CRISP	148	695.3	7016	1	0	1.95	4.54	1.008	5.8%	3.2%	0.8%
nb4, 50%	mPL6	256	295.7	196924	35	42	4.62	12.94	1.039	—	—	—
	mPL6 + Bonn	522	300.7	263756	>1440	5380	4.89	14.12	1.134	-5.7%	-9.1%	-9.5%
	mPL6 + CRISP	388	293.1	130480	4	0	4.25	12.41	1.028	8.1%	4.1%	1.1%
nb5, 50%	mPL6	435	528.6	92014	>1440	1440	8.17	23.44	1.023	—	—	—
	mPL6 + Bonn	883	525.1	73026	9	0	8.12	23.14	1.012	0.5%	1.3%	1.1%
	mPL6 + CRISP	770	521.3	16838	4	0	7.53	22.42	1.007	7.8%	4.3%	1.6%
nb6, 90%	mPL6	359	499.2	479612	>1440	24488	8.80	19.14	1.129	—	—	—
	mPL6 + Bonn	747	503.9	382422	>1440	702	9.05	19.93	1.176	-2.9%	-4.2%	-4.7%
	mPL6 + CRISP	449	502.8	355260	400	0	8.45	10.08	1.091	3.9%	3.2%	3.8%
nb7, 70%	mPL6	1020	1101	100578	46	250	16.37	31.73	1.014	—	—	—
	mPL6 + Bonn	2056	1105	76864	25	60	16.36	31.74	1.012	0.1%	0.0%	0.2%
	mPL6 + CRISP†	1306	1078	31940	5	0	14.58	29.50	1.008	10.9%	7.0%	0.6%

experiments because we found that it is trivially unroutable: it contains a standard cell (`o389042`) which connects to over 2200 nets at the same pin. No global routing GCell in `newblue3` has capacity for so many nets, making overflow-free global routing impossible.

Table 9.2: Using ECO-system [117], the Bonn flow [16] and CRISP to improve the routability of routable mPL6 [28] placements of ISPD contest benchmarks [105, 106]. Detouring is measured as the ratio of global routing segments to FLUTE [44] Steiner wirelength.

Benchmark & Target density	Placement			Global routing (NTHU-Route 2.0 [33])						Reduction of		
	Flow	Runtime (min)	HPWL (e6)	Estimated overflow	Runtime (min)	Final overflow	Vias (e6)	RWL (e6)	Detour ratio	Vias	RWL	Detours
ad1, 70%	mPL6	68	84.7	124186	12	0	1.79	4.62	1.061	—	—	—
	mPL6 + ECO-system	146	260.0	8903538	>1440	8416374	1.84	12.12	1.316	-3.2%	-162%	-25.5%
	mPL6 + Bonn	134	86.0	105482	5	0	1.78	4.62	1.045	0.2%	0.1%	1.6%
	mPL6 + CRISP	97	85.3	100784	5	0	1.72	4.54	1.045	1.8%	3.6%	1.6%
ad2, 60%	mPL6	89	103.2	41322	1	0	1.92	5.29	1.025	—	—	—
	mPL6 + ECO-system	253	312.1	6264548	>1440	4216896	2.93	16.50	1.422	-52.6%	-212%	-39.7%
	mPL6 + Bonn	179	107.7	28734	1	0	1.92	5.43	1.020	0.1%	-2.5%	0.5%
	mPL6 + CRISP	113	100.7	27174	1	0	1.80	5.08	1.021	6.3%	4.0%	0.4%
ad3, 70%	mPL6	305	226.9	196600	13	0	3.82	12.37	1.037	—	—	—
	mPL6 + Bonn	614	228.7	167068	6	0	3.82	12.41	1.033	0.1%	-0.3%	0.4%
	mPL6 + CRISP	391	224.4	107830	4	0	3.54	11.92	1.027	8.8%	5.1%	1.0%
ad4, 80%	mPL6	252	194.8	57474	4	0	3.44	10.50	1.013	—	—	—
	mPL6 + Bonn	533	195.3	47510	4	6	3.43	10.50	1.012	0.2%	0.0%	0.1%
	mPL6 + CRISP	359	195.9	9804	2	0	3.14	10.21	1.008	8.7%	2.8%	0.5%
ad5, 60%	mPL6	414	391.2	85008	5	0	5.55	14.21	1.017	—	—	—
	mPL6 + Bonn	777	396.7	75314	3	0	5.56	14.34	1.016	-0.2%	-0.9%	0.1%
	mPL6 + CRISP	584	385.0	28892	2	0	5.08	13.57	1.013	8.5%	4.5%	0.4%
bb1, 50%	mPL6	71	121.5	76468	5	0	1.90	4.70	1.042	—	—	—
	mPL6 + Bonn	150	123.5	72548	3	0	1.90	4.74	1.041	-0.2%	-0.9%	0.1%
	mPL6 + CRISP	123	124.8	50442	2	0	1.79	4.62	1.026	5.5%	1.6%	1.6%
bb2, 40%	mPL6	212	186.7	32734	4	0	4.04	9.06	1.020	—	—	—
	mPL6 + Bonn	400	189.7	30766	2	0	4.04	9.50	1.020	-0.1%	-0.9%	0.0%
	mPL6 + CRISP	357	183.4	14296	2	0	3.69	8.98	1.016	8.7%	4.6%	0.4%
bb3, 100%	mPL6	258	344.1	21132	2	0	6.07	13.53	1.012	—	—	—
	mPL6 + Bonn	507	345.2	16522	2	0	6.07	13.55	1.011	-0.1%	-0.2%	0.1%
	mPL6 + CRISP	348	343.3	8838	2	0	5.83	13.26	1.008	3.8%	1.9%	0.7%
nb2, 100%	mPL6	177	199.4	6390	1	0	3.28	7.67	1.006	—	—	—
	mPL6 + Bonn	372	198.7	5836	1	0	3.28	7.66	1.006	-0.1%	0.2%	0.0%
	mPL6 + CRISP	253	198.3	2620	1	0	3.08	7.45	1.006	6.0%	2.8%	0.1%
nb6, 80%	mPL6	368	515.2	254300	23	0	8.50	18.30	1.036	—	—	—
	mPL6 + Bonn	779	519.8	201594	14	0	8.49	18.31	1.029	0.2%	-0.1%	0.7%
	mPL6 + CRISP	505	515.0	132644	8	0	7.74	17.41	1.022	8.9%	4.8%	1.4%
nb7, 60%	mPL6	1105	1150	48522	10	2	16.14	32.11	1.008	—	—	—
	mPL6 + Bonn	2134	1154	36770	10	0	16.13	32.15	1.007	0.1%	-0.1%	0.1%
	mPL6 + CRISP	1597	1125	4402	4	0	14.41	29.95	1.003	10.8%	6.7%	0.5%

Routability evaluation. To determine routability and guide CRISP, we use the winning router of the ISPD 2008 routing contest, NTHU-Route 2.0 [33]. CRISP limits NTHU-Route 2.0 to a single round of rip-up and reroute. This limits the runtime of the router as well as the detouring in the routed solution.[1] Final routability of each placement is determined by routing the nets using NTHU-Route 2.0 with default parameters.

[1]The initial routing produced by NTHU-Route 2.0 uses Steiner trees, so at least one round of rip-up and reroute must be performed for detouring to occur.

Comparing routability improvement techniques. We compare CRISP against state-of-the-art congestion reduction techniques ECO-system and a congestion reduction flow based on BonnPlace [16] that we refer to as the "Bonn flow." ECO-system is an incremental placement technique which, when in congestion reduction mode, estimates congestion with a probabilistic congestion map and re-allocates whitespace to routing congested regions of the placement. The Bonn flow, similar to CRISP, temporarily inflates standard cells in routing congested regions. The Bonn flow differs from CRISP in that inflates all cells in regions where routing resources are more than 100% utilized. The techniques employed by BonnPlace are not strictly incremental since they are presented for use inside a global placer, so we make the Bonn flow applicable to any initial placement by first estimating congestion with NTHU-Route 2.0, inflating all cells in congested regions, and then replacing the design with mPL6 using the same target density as the initial placement.

Tables 9.1 and 9.2 compare CRISP to ECO-system and the Bonn flow on unroutable and routable placements of the ISPD benchmarks, respectively. For both the routable and unroutable benchmarks, ECO-system placements were in one case illegal and in other cases uncompetitive. We believe that this is caused by the use of probabilistic congestion maps which cause ECO-system to think that areas near routing blockages have the same routing resources as other areas. Thus many standard cells are placed on top of blockages, leaving a large burden to the legalizer of ECO-system. Indeed more than 80% of ECO-system's runtime was spent during legalization. For these reasons, we only provide results for ECO-system on `adaptec1` and `adaptec2`. For unroutable benchmarks, the Bonn flow improves via counts by 1.4%, routed wirelength by 1.6% and detouring by 3.3% on

average. On the same designs, CRISP improves via counts by 8.7%, routed wirelength by 6.5% and detouring by 5.3% on average. CRISP improves the routability of each of the unroutable designs whereas the Bonn flow degrades routability on `newblue4`. Of the 13 unroutable benchmarks, CRISP produces the best solutions on 11 of the 13, with the Bonn flow producing less routing overflow on `adaptec1` and `adaptec2`.

Results on the 11 ISPD contest benchmarks where we could produce routable solutions using mPL6 alone are given in Table 9.2. The Bonn flow reduces via counts and detouring by 0.1% and 0.5%, respectively, but *degrades* routability on `adaptec4` and *increases* routed wirelength by 0.7%. CRISP improves routability on all test cases, and decreases via counts by 6.8%, routed wirelength by 4.0% and detouring by 0.8%. CRISP takes 38% of the runtime of placing the design from scratch and is faster than the Bonn flow in all cases.

9.3.2 Commercial Designs

Timing impact of CRISP. To judge how effective CRISP is at preserving the timing characteristics of commercial designs, we added CRISP to an industrial physical-synthesis flow. We applied CRISP to four designs which have high congestion after the initial placement stage of the flow. After CRISP, we applied medium effort timing transformations to optimize critical paths as well as the timing histogram. These transformations mainly consist of buffering and resizing techniques. After these timing optimizations, we measured timing with an industry timer and report the results in Table 9.3. Designs 1 and 2 in Table 9.3 are large commercial designs with approximately 1,000,000 objects (20,000 fixed blockages) and 700,000 objects (90,000 fixed blockages), respectively. Designs 3 and 4

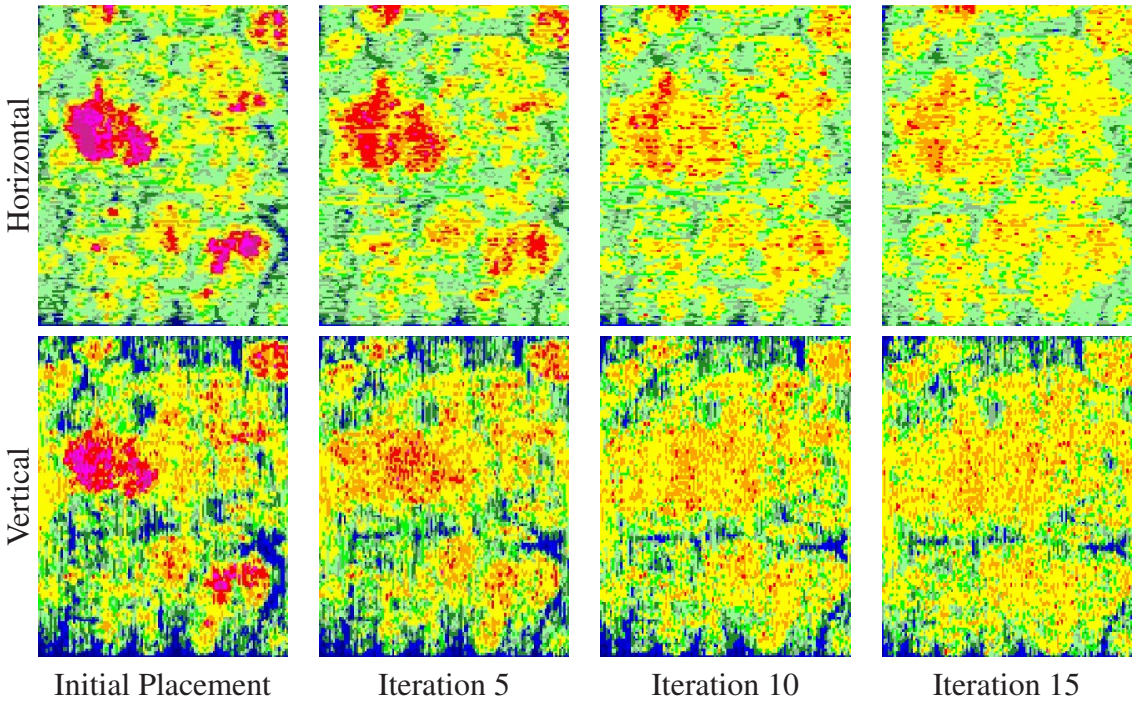


Figure 9.5: Incrementally relieving congestion problems on a heavily congested industrial design with low whitespace. Areas colored pink and purple have global routing resource usage over 100%. These areas are targeted by CRISP and eliminated.

are smaller designs both with approximately 100,000 objects (1,000 fixed blockages). For each design, we report congestion and timing statistics after timing optimization has been performed. For all four designs, CRISP is very effective in reducing the number of nets with at least 90% and 100% congestion. In terms of timing, CRISP has better worst slack in three of the four layouts and better total negative slack in the two larger designs, with only minor degradations for the smaller designs. Since CRISP does not consider timing in its flow, we attribute the gains in timing to the fact that the placements are more spread and it is easier to apply cell resizing and buffering to them.

Detailed routing improvement. To judge the effectiveness of pin-density congestion removal by CRISP on detailed routing, we chose 40 high-performance designs and ran them through an industrial physical-synthesis flow. We added CRISP to the flow after

Table 9.3: CRISP's global routing and timing impact on commercial designs. For congestion we report the percentage of nets which are at least 90% and 100% congested. Fewer nets congested implies better routability.

Design	CRISP used?	Routing congestion		Timing slack (ns)	
		100% nets	90% nets	Worst	Total negative
design 1	No	18.1%	19.8%	-4.861	-177020
	Yes	**0.4%**	**6.8%**	**-3.292**	**-175931**
design 2	No	0.7%	4.7%	-0.961	-1249
	Yes	**0.2%**	**2.3%**	**-0.904**	**-1187**
design 3	No	4.3%	9.6%	**0.096**	0
	Yes	**0.1%**	**1.5%**	0.072	0
design 4	No	5.7%	11.7%	-0.097	**-17.8**
	Yes	**0.1%**	**3.2%**	**-0.062**	-18.6

clocks were inserted into the design such that CRISP targeted only pin density. We report results from a few of the designs as well as a summary of the designs in Table 9.4. On average, CRISP was able to reduce detailed routing runtime by 10.2%, detoured nets by 4.5%, DRC violations by 79.0% and shorts & opens by 62.5%. CRISP increased DRC violations, shorts or open count in five of the 40 designs, but by only one violation, short or open in these designs.

Core area reduction. Previous work has optimized routability of designs in order to reduce routing violations, routed wirelength and turn-around-time. While these are important metrics which we evaluate in our experiments, they do not necessarily communicate all the benefits that strong place-and-route tools can provide such as the ability to reduce manufacturing cost. To this end, we worked with expert designers to re-floorplan design 3 from Table 9.3 to use less die area and fewer routing resources. The result is design 4 (also shown in Table 9.3), which uses 5% less area than design 3. This increased the design utilization from 73% to 79%, which is high for a modern design. This also provides less area with which to perform spreading during CRISP. As Table 9.3 shows, without CRISP

Table 9.4: Impact of pin-density CRISP on high-performance commercial designs.

Design	CRISP used?	Detailed routing runtime (m)	Detoured nets	DRC violations	Shorts & Opens
design 5	No	357	255	5	4
	Yes	122	227	0	0
design 6	No	11	69	0	0
	Yes	11	60	0	0
design 7	No	81	59	0	0
	Yes	83	41	0	0
design 8	No	73	9	0	0
	Yes	57	7	0	0
Avg. improvement		10.2%	4.5%	79.0%	62.5%

design 4 would have been extremely difficult to route since 5.7% of its nets were 100% or more congested after timing optimization. The congestion of design 4 during CRISP is shown in Figure 9.5. After applying CRISP to make design 4 routable, we used industry timing optimizations to close on timing and inserted clocks. After clock insertion, we used CRISP again to eliminate areas of high pin density which reduced shorts and opens from 370 to 41 and detailed routing runtime from 10.9 hours to 7.4 hours. The designers were able to fix all shorts and opens with some minor alterations after CRISP, making design 4 routable without violations.

9.4 Conclusions

In this chapter we have presented CRISP, an incremental technique for Congestion Reduction by Iterated Spreading during Placement. CRISP combines highly accurate congestion modeling with carefully chosen incremental placement transformations. CRISP leverages recent advances in global routing algorithms to model congestion and enhancing previous congestion-driven placement techniques to make them incremental. We have empirically validated CRISP on a number of modern placement instances using (i) aca-

demic tools and (ii) integrated industrial design-tool flows. CRISP consistently improves routability on common benchmarks, reducing via counts by 8.7%, global routed wirelength by 6.5% and detouring by 5.3%. We have also verified CRISP's effectiveness on industrial designs and demonstrated CRISP's ability to preserve timing and improve detailed routability by eliminating pin-density hot-spots. Finally, we have shown that with the aid of strong place-and-route tools, designers can shrink die sizes, which leads to savings in manufacturing cost. We believe that our work is the first to demonstrate the link between improved congestion-driven placement and manufacturing cost reductions.

PART V

Summary

CHAPTER X

Conclusions and Future Work

Modern designs comprise millions of standard cells, macros and signal nets, and it is predicted that their size and complexity will continue to increase in the near term. Such designs require powerful and scalable placement and routing techniques to reach design closure. Rather than crudely approximate design goals, state-of-the-are techniques must faithfully model key layout characteristics and solve important problems in physical design, such as ensuring routability and meeting timing constraints.

In this dissertation, we have identified new objectives, constraints and concerns in VLSI physical design, and developed new computational techniques to address them. Specifically, we developed new techniques for VLSI placement and routing to improve solution quality, robustness and make optimizations more consistent with each other in

a modern industrial physical synthesis flow. Below we summarize our contributions and discuss avenues for future work.

10.1 Summary of Contributions

We have found that a significant source of suboptimality in both academic and industry physical design tools today is the fact that they optimize an incorrect or wrong objective function. One glaring example of this phenomenon is the fact that global and detailed placement algorithms target the minimization of half-perimeter wirelength rather than more relevant objectives such as final routed wirelength, timing of critical paths, manufacturability, etc. These discrepancies are especially important as designs are produced at sub-65nm technologies because the suboptimality is amplified. To this end, we identify new objectives, constraints and concerns in physical design, and develop new algorithms to solve them.

Part II introduces our work on VLSI placement, starting with techniques for accurate control of whitespace allocation. These techniques can suit a wide range of important objectives at the nanometer scale such as routability, yield and manufacturability. We also introduce the first work in the literature for minimizing Steiner wirelength in global placement. In combination with our routability-driven whitespace allocation techniques, our ROOSTER placer outperforms all previous literature in via counts on a wide variety of publicly available benchmarks in addition to highly competitive routed wirelength. Lastly, we detail our incremental placement techniques, collectively known as ECO-system. Incremental placement is vital to preserve design properties across optimizations, Engineering Change Orders (ECOs) and design iterations, especially at the nanometer scale. ECO-

system provides fast and robust legalization for a wide range of design modifications from high-level synthesis to physical synthesis and detailed placement. ECO-system detects geometric regions and sections of the netlist that require modification and applies an adequate amount of change in each case. This allows ECO-system to run many times faster than a global placer, increase wirelength only slightly, and have minimal impact on timing.

Part III presents our FGR global routing framework. FGR (Fairly Good Router) utilizes Discrete Lagrange Multipliers (DLM) and A*-search based maze routing to support two- and three-dimensional routing of ASICs with millions of nets. FGR outperforms the best results from the ISPD '07 Global Routing Contest, as well as previous literature, in terms of route completion, runtime and total wirelength. We also present enhancements to the FGR framework in terms of runtime and solution quality. These improvements increase the applicability of FGR as a fast and accurate routability predictor early in the design flow. We also show how to make FGR optimize the length of some nets preferentially over others, making FGR useful for timing-driven routing.

Our work culminates in Part IV with the integration of our routing techniques into several modern design flows. We present CRISP (Congestion Reduction by Iterated Spreading during Placement) which is an incremental global and detailed routability improvement technique. CRISP improves the global and detailed routability of a given placement while preserving relevant statistics, such as timing, which may already have been optimized in a design. CRISP identifies areas of congestion in a design by calling a global router and limiting its detouring. This allows CRISP to account for areas of congestion caused by fixed routing obstacles as well as those that arise due to detouring. CRISP also identifies

regions of a design that may prove difficult to a detailed router by actively reducing areas of high pin-density throughout the design. We show that CRISP improves the routability of both academic and industry placements and allows an experienced designer to *reduce manufacturing costs by shrinking the floorplan of an industrial design*. This is the first published demonstration of how enhancing place-and-route techniques can directly improve manufacturing cost of commercial multi-million-gate designs.

10.2 Directions for Future Work

There are several fronts on which to continue our work. One direction for future research is further extending the FGR routing framework. For example, built-in support for incremental routing would be a useful addition. Like incremental placement, incremental routing is necessary in a modern design flow to preserve good solution quality and design properties and keep design iteration times small. This would improve the scalability of algorithms such as CRISP as well.

Technology-related concerns, such as pitch- and parasitics-aware layer assignment, would be valuable extensions to FGR. It is becoming more common for different layers of metal to have drastically different timing properties in modern designs. For example, wires in the highest metal layers are usually thicker and faster, but are therefore a more scarce resource. Current global routing formulations and benchmarks do not take this phenomenon into account; indeed, they assume that wire thickness is the same at all layers of the routing grid. Furthermore, individual nets can be assigned to one or more layers for timing reasons, inducing layer constraints which invalidate the assumptions of most if not all layer assignment techniques currently in the literature.

Timing-driven enhancements to FGR are also a possibility. In Chapter VII we showed that net weights can be used to reduce the length of specific nets. The dual Lagrange formulation presented in Section 8.3.2 can also be used to place length limits on individual nets. Combining these two routing techniques with net criticality information and critical-net weighting, used in timing-driven placement flows, FGR could generate timing-driven global routing solutions.

Yield is significantly impacted by wire shorts and wire opens. The susceptibility to these random defects is measured in terms of critical area (with respect to each type of defect), i.e., the area where a particle of a certain size will cause a given defect. Susceptibility to opens is a function of wirelength, whereas susceptibility to shorts is a function of wire density. Indiscriminately spreading wires further apart will in many cases increase their length, therefore yield optimization for shorts and opens must be carefully balanced. FGR could be extended to accomplish such optimization by tracking wire density during the global routing process and estimating critical areas based on wirelength and wire density.

Our CRISP work pointed out that there is a discrepancy between global and detailed routability of a design. A design may quickly admit violation-free global routes, but local pin-access problems may block detail routes. CRISP was able to mitigate this problem by examining and eliminating pin-density congestion in the design. This drastically reduced the detailed routing problems, the remainder of which were corrected by a skilled chip-designer at IBM. These sorts of failures require more in-depth study to determine how they might be detected earlier (perhaps by a global router) and fixed with reduced designer effort.

We also suggest to integrate more global routing ideas into placement to close the gap between these separate stages in modern design flows. CRISP effectively improves the routability of designs by reducing routed wirelength, via counts and detouring, but does not currently leverage all the information given to it by the global router. CRISP can be extended to target detoured nets for optimization directly. Specifically, CRISP can identify those nets which are detoured by a global router and find ways reduce their detouring by giving these nets more importance during global placement. Routing-aware detailed placement techniques can also be introduced. For example, a detailed placement engine can (i) focus on those nets which use many vias and (ii) see if via counts can be decreased through pin alignment moves. Combined with a fast incremental router, detailed placement techniques would be able to accurately assess the impact of local moves on routability and routed wirelength. These techniques will produce more routable placements and help placement software optimize for additional relevant objectives such as timing.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa and I. L. Markov,"Unification of Partitioning, Placement and Floorplanning," *ICCAD*, pp. 550-557, 2004.

[2] S. N. Adya and I. L. Markov, "Consistent Placement of Macro-blocks Using Floorplanning and Standard-Cell Placement," *ISPD*, pp. 12-17, 2002.

[3] S. N. Adya and I. L. Markov,"Fixed-outline Floorplanning: Enabling Hierarchical Design," *IEEE Trans. on VLSI*, vol. 11, no. 6, pp. 1120-1135, 2003. (*ICCD* 2001, pp. 328-334).

[4] S. N. Adya and I. L. Markov, "Combinatorial Techniques for Mixed-size Placement," *ACM Trans. on Design Autom. of Elec. Sys.*, vol. 10, no. 5, 2005. (*ISPD* 2002, pp. 12-17).

[5] S. N. Adya, I. L. Markov and P. G. Villarrubia, "On Whitespace and Stability in Physical Synthesis," *Integration: the VLSI Journal*, vol. 25, no. 4, pp. 340-362, 2006. (*ICCAD* 2003, pp. 311-318).

[6] A. Agnihotri et al., "Mixed Block Placement via Fractional Cut Recursive Bisection," *IEEE Trans. Computer-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 5, pp 748-761, 2005. (*ICCAD*, pp. 307-310, 2003).

[7] C. Albrecht, "Global Routing by New Approximations for Multicommodity Flow," *IEEE Trans. on CAD*, vol. 20, no. 5, pp. 622-632, 2001.

[8] C. J. Alpert, A. Devgan and C. Kashyap, "A Two Moment RC Delay Metric for Performance Optimization," *ISPD*, pp. 69-74, 2000.

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

[10] C. J. Alpert, G.-J. Nam and P. G. Villarrubia, "Effective Free Space Management for Cut-Based Placement via Analytical Constraint Generation," *IEEE Trans. on CAD*, vol. 22, no. 10, pp. 1343-1353, 2003. (*ICCAD* 2002, pp. 746-751).

[11] C. J. Alpert, G.-J. Nam, P. Villarrubia and M. C. Yildiz, "Placement Stability Metrics," *ASP-DAC*, pp. 1144-1147, 2005.

[12] C. J. Alpert et al., "Techniques for Fast Physical Synthesis," *Proc. of the IEEE* 95(3), pp. 573-598, 2007.

[13] P. Azzoni, M. Bertoletti, N. Dragone, F. Fummi, C. Guardiani and W. Vendraminetto, "Yield-aware Placement Optimization," *DATE*, pp. 1232-1237, 2007.

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

[15] E. Bozorgzadeh, R. Kastner and M. Sarrafzadeh, "Creating and Exploiting Flexibility in Rectilinear Steiner Trees," *IEEE Trans. on CAD*, vol. 22, no. 5, pp. 605-615, 2003.

[16] U. Brenner and A. Rohe, "An Effective Congestion Driven Placement Framework," *IEEE Trans. on CAD*, vol. 22, no. 4, pp. 387-394, 2003. (*ISPD* 2002, pp. 6-11).

[17] U. Brenner and J. Vygen, "Faster Optimal Single-Row Placement with Fixed Ordering," *DATE* pp. 117-121, 2000.

[18] U. Brenner and J. Vygen, "Legalizing a Placement With Minimum Total Movement," *IEEE Trans. on CAD*, vol. 23, no. 12, pp. 1597-1613, 2004. (*ISPD* 2004, pp. 2-9).

[19] M. Breuer, "Min-cut Placement," *Journal of Design Automation and Fault Tolerant Computing*, vol. 1, no. 4, pp. 343-362, 1977. (*DAC* 1977, pp. 284-290).

[20] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wirelength Estimations for Row-Based Placement," *IEEE Trans. on CAD*, vol. 18, no. 9, pp. 1265-1278, 1999.

[21] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?" *DAC*, pp. 477-482, 2000.

[22] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Design and Implementation of Move-based Heuristics for VLSI Hypergraph Partitioning," *ACM J. of Experimental Algorithms*, vol. 5, 2000.

[23] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-cell Layout," *IEEE Trans. on CAD*, vol. 19, no. 11, pp. 1304-1314, 2000. (*ISPD* 1999, pp. 90-96).

[24] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Hierarchical Whitespace Allocation in Top-down Placement," *IEEE Trans. on CAD*, vol. 22, no. 11, pp. 716-724, 2003.

[25] H. H. Chan, S. N. Adya and I. L. Markov, "Are Floorplan Representations Useful in Digital Design?" *ISPD*, pp. 129-136, 2005.

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

[27] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze and M. Xie, "mPL6: Enhanced Multilevel Mixed-size Placement," *ISPD*, pp. 212-214, 2006.

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

[29] C.-C. Chang, J. Cong, D. Pan and X. Yuan, "Multilevel Global Placement with Congestion Control," *IEEE Trans. Computer-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 4, pp. 395-409, 2003.

[30] C.-C. Chang, J. Cong, M. Romesis and M. Xie, "Optimality and Scalability Study of Existing Placement Algorithms," *IEEE Trans. on CAD*, pp. 537-549, 2004.

[31] C.-C. Chang, J. Cong and X. Yuan, "Multi-Level Placement for Large-Scale Mixed-Size IC Designs," *ASP-DAC*, pp. 325-330, 2003.

[32] Y. C. Chang et al., "B*-trees: A New Representation for Non-Slicing Floorplans," *DAC*, pp. 458-463, 2000.

[33] Y.-J. Chang, Y.-T. Lee and T.-C. Wang, "NTHU-Route 2.0: A fast and stable global router," *ICCAD* pp. 338-343, 2008.

[34] H.-Y. Chen, M.-F. Chiang, Y.-W. Chang, L. Chen and B. Han, "Novel Full-chip Gridless Routing Considering Double-via Insertion," *DAC*, pp. 755-760, 2006.

[35] T.-C. Chen, M. Cho, D. Z. Pan and Y.-W. Chang, "Metal-Density Driven Placement for CMP Variation and Routability," *ISPD*, pp. 31-38, 2008.

[36] T.-C. Chen and Y.-W. Chang, "Modern Floorplanning Based on Fast Simulated Annealing," *ISPD*, pp. 104-112, 2005.

[37] T.-C. Chen, Y.-W. Chang and S.-C. Lin, "IMF: Interconnect-Driven Multilevel Floorplanning for Large-Scale Building-Module Designs," *ICCAD*, pp. 159-164, 2005.

[38] T.-C. Chen, Y.-W. Chang and S.-C. Lin, "A Novel Framework for Multilevel Full-chip Gridless Routing," *ASP-DAC*, pp. 636-641, 2006.

[39] C.-L. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling," *ICCAD*, pp. 690-695, 1994.

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

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

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

[43] C. Chiang and J. Kawa, "Design for Manufacturability and Yield for Nano-Scale CMOS," Springer, 2007.

[44] C. C. N. Chu and Y.-C. Wong, "FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design," *IEEE Trans. on CAD*, vol. 27, no. 1, pp. 70-83, 2008. `http://class.ee.iastate.edu/cnchu/flute.html`

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

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

[47] J. Cong and G. Luo, "Highly Efficient Gradient Computation for Density-Constrained Analytical Placement Methods," *ISPD*, pp. 39-46, 2008.

[48] J. Cong, G. Nataneli, M. Romesis and J. Shinnerl, "An Area-optimiality Study of Floorplanning," *ISPD*, pp. 78-83, 2004.

[49] J. Cong, M. Romesis and J. Shinnerl, "Fast Floorplanning by Look-Ahead Enabled Recursive Bipartitioning," *ASP-DAC*, pp. 1119-1122, 2005.

[50] J. Cong, M. Romesis and J. Shinnerl, "Robust Mixed-Size Placement Under Tight White-Space Constraints," *ICCAD*, pp. 165-172, 2005.

[51] J. Cong and M. Sarrafzadeh, "Incremental Physical Design," in *ISPD*, pp. 84-92, 2000.

[52] J. Cong and M. Xie, "A Robust Detailed Placement for Mixed-Size IC Designs," *ASP-DAC*, pp. 188-194, 2006.

[53] J. Cong, M. Xie and Y. Zhang, "MARS—A Multilevel Full-Chip Gridless Routing System," *IEEE Trans. on CAD*, vol. 24, no. 3, pp. 382-394, 2005.

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

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

[56] K. Doll, F. M. Johannes and K. J. Antreich, "Iterative Placement Improvement By Network Flow Methods," *IEEE Trans. Computer-Aided Des. Integr. Circuits Syst.*, vol. 13, no. 10, pp. 1189-1200, 1994.

[57] W. Donath et al., "Transformational Placement and Synthesis", *DATE*, pp. 194-201, 2000.

[58] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard Cell VLSI Circuits," *IEEE Trans. on CAD*, vol. 4, no. 1, pp. 92-98, 1985.

[59] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *DAC*, pp. 175-181, 1982.

[60] M. R. Garey and D. S. Johnson, "The Rectilinear Steiner Problem is NP-Complete," *SIAM Journal of Applied Mathematics*, vol. 32, pp. 826-834, 1977.

[61] R. Goering, "Cadence CTO: CAD 'Foundations' Must Change," EETimes, April 11, 2006, `http://www.eetimes.com/showArticle.jhtml?articleID=185300099`

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

[63] A. V. Goldberg, "An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm," *J. Algorithms*, vol. 22, no. 1, pp. 1-29, 1997.

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

[65] D. Hill, "Method and System for High Speed Detailed Placement of Cells Within an Integrated Circuit Design," *U.S. Patent 6370673*, April 2002.

[66] S. Hu and J. Hu, "Pattern Sensitive Placement For Manufacturability," *ISPD*, pp. 27-34, 2007.

[67] J. Hu and S. S. Sapatnekar, "A Survey on Multi-net Global Routing for Integrated Circuits," *Integration, the VLSI Journal*, vol. 31, no. 1, pp. 1-49, 2001.

[68] D. J.-H. Huang, and A. B. Kahng, "Partitioning-based Standard-cell Global Placement With an Exact Objective," *ISPD*, pp. 18-25, 1997.

[69] S. W. Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement," *ICCAD*, pp. 165-170, 2000.

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

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

[72] IWLS 2005 Benchmarks, `http://iwls.org/iwls2005/benchmarks.html`

[73] D. Jariwala and J. Lillis, "On Interactions Between Routing and Detailed Placement," *ICCAD*, pp. 387-393, 2004.

[74] Z.-W. Jiang et al., "NTUPlace2: A Hybrid Placer Using Partitioning and Analytical Techniques," *ISPD*, pp. 215-217, 2006.

[75] Z.-W. Jiang, B.-Y. Su and Y.-W. Chang, "Routability-driven analytical placement by net overlapping removal for large-scale mixed-size designs," *DAC*, pp. 167-172, 2008.

[76] A. B. Kahng and S. Mantik, "On Mismatches Between Incremental Optimizers and Instance Perturbations in Physical Design Tools," *ICCAD*, pp. 17-22, 2000.

[77] A. B. Kahng, S. Mantik and I. L. Markov, "Min-max Placement For Large-scale Timing Optimization," *ISPD*, pp. 143-148, 2002.

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

[79] A. B. Kahng and S. Reda, "Placement Feedback: A Concept and Method for Better Min-cut Placement," *DAC*, pp. 357-362, 2004.

[80] A. B. Kahng and S. Reda, "Evaluation of Placer Suboptimality Via Zero-Change Netlist Transformations," *ISPD*, pp. 208-215, 2005.

[81] A. B. Kahng and G. Robins, "A New Class of Iterative Steiner Tree Heuristics With Good Performance," *IEEE Trans. on CAD*, vol. 11, no. 7, pp. 893-902, 1992.

[82] A. B. Kahng and Q. Wang, "Implementation and Extensibility of an Analytic Placer," *IEEE Trans. on CAD*, vol. 25, no. 5, pp. 734-747, 2005.

[83] A. B. Kahng and X. Xu, "Accurate Pseudo-constructive Wirelength and Congestion Estimation," *SLIP*, pp. 81-86, 2003.

[84] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Domain," *IEEE TVLSI*, vol. 7, no. 1, pp. 69-79, 1999.

[85] R. Kastner, E. Bozorgzadeh and M. Sarrafzadeh, "Pattern Routing: Use and Theory for Increasing Predictability and Avoiding Coupling," *IEEE Trans. on CAD*, vol. 21, no. 7, pp. 777-790, 2002.

[86] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh and P. H. Madden, "Recursive Bisection Based Mixed Block Placement," *ISPD*, pp. 84-89, 2004.

[87] J. Kim, M. C. Papaefthymiou and J. L. Neves, "Parallelizing Post-Placement Timing Optimization," *IPDPS* 2006.

[88] S. H. Kulkarni, D. Sylvester and D. Blaauw, "A Statistical Framework for Post-silicon Tuning Through Body Bias Clustering," *ICCAD*, pp. 39-46, 2006.

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

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

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

[92] C. Li, C.-K. Koh and P. H. Madden, "Floorplan Management: Incremental Placement for Gate Sizing and Buffer Insertion," *ASP-DAC*, pp. 349-354, 2005.

[93] C. Li, M. Xie, C.-K. Koh, J. Cong and P. H. Madden, "Routability-driven Placement and White Space Allocation," *IEEE Trans. on CAD*, vol. 26, no. 5, pp. 858-871, 2007.

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

[95] L. Luo, Q. Zhou, X. Hong and H. Zhou, "Multi-stage Detailed Placement Algorithm for Large-Scale Mixed-Mode Layout Design," *ICCSA*, pp. 896-905, 2005.

[96] T. Luo, H. Ren, C. J. Alpert and D. Pan, "Computational Geometry Based Placement Migration," *ICCAD*, pp. 41-47, 2005.

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

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

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

[100] M. D. Moffitt, A. N. Ng, I. L. Markov and M. E. Pollack, "Constraint-driven Floorplan Repair," *DAC*, pp. 1103-1108, 2006.

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

[102] G.-J. Nam, "ISPD 2006 Placement Contest: Benchmark Suite and Results," *ISPD*, p. 167, 2006.

[103] G.-J. Nam, F. Aloul, K. A. Sakallah and R. A. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constaints," *IEEE Trans. on Computers*, vol. 53, no. 6, pp. 688-696, 2004.

[104] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter and M. Yildiz, "The ISPD2005 Placement Contest and Benchmark Suite," *ISPD*, pp. 216-220, 2005.

[105] G.-J. Nam, C. C. N. Sze and M. Can Yildiz, "The ISPD global routing benchmark suite," *ISPD* pp. 156-159, 2008.

[106] G.-J. Nam, M. C. Yildiz, D. Z. Pan and P. H. Madden: "ISPD placement contest updates and ISPD 2007 global routing contest," *ISPD*, p. 167, 2007.

[107] A. N. Ng, I. Markov, R. Aggarwal and V. Ramachandran, "Solving Hard Instances of Floorplacement," *ISPD*, pp. 170-177, 2006.

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

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

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

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

[112] S. Reda and A. Chowdhary, "Effective Linear Programming Based Placement Methods," *ISPD*, pp. 186-191, 2006.

[113] H. Ren, D. Z. Pan, C. J. Alpert and P. Villarrubia, "Diffusion-based Placement Migration," *DAC*, pp. 515-520, 2005.

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

[115] J. A. Roy, S. N. Adya, D. A. Papa and I. L. Markov, "Min-cut Floorplacement," *IEEE Trans. on CAD*, vol. 25, no. 7, pp. 1313-1326, 2006. (*ICCAD* 2004, pp. 550-557).

[116] J. A. Roy and I. L. Markov, "Seeing the Forest and the Trees: Steiner Wirelength Optimization in Placement," *IEEE Trans. on CAD*, vol. 26, no. 4, pp. 632-644, 2007 (*ISPD* 2006, pp. 78-85).

[117] J. A. Roy and I. L. Markov, "ECO-system: Embracing the Change in Placement," *IEEE Trans. on CAD*, vol. 26, no. 12, pp. 2173-2185, 2007 (*ASP-DAC* 2007, pp. 147-152).

[118] J. A. Roy, A. N. Ng, R. Aggarwal, V. Ramachandran and I. L. Markov, "Solving Modern Mixed-size Placement Instances," to appear in *Integration, the VLSI Journal*, 2009.

[119] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, J. F. Lu, A. N. Ng and I. L. Markov, "Capo: Robust and Scalable Open-Source Min-cut Floorplacer," *ISPD*, pp. 224-227, 2005.

[120] J. A. Roy, D. A. Papa, A. N. Ng and I. L Markov, "Satisfying Whitespace Requirements in Top-down Placement," *ISPD*, pp. 206-208, 2006.

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

[122] L. K. Scheffer, L. Lavagno and G. Martin, eds., *Electronic Design Automation for Integrated Circuits Handbook*, Chapter 8: Routing, CRC Press, 2006.

[123] N. Selvakkumaran and G. Karypis, "THETO - A Fast and High-quality Partitioning Driven Global Placer," Technical Report 03-046, Univ. of Minnesota, 2003.

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

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

[126] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," *DATE*, pp. 1226-1231, 2007.

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

[128] P. R. Suaris and G. Kedem, "An Algorithm for Quadrisection and Its Application to Standard Cell Placement," *IEEE Trans. on Circuits and Systems*, vol. 35, no. 3, pp. 294-303, 1988. (*ICCAD* 1987, pp. 474-477).

[129] T. Taghavi, X. Yang, B.-K. Choi, M. Wang and M. Sarrafzadeh "Dragon2005: Large-Scale Mixed-size Placement Tool," *ISPD*, pp. 245-247, 2005.

[130] T. Taghavi, X. Yang, B.-K. Choi, M. Wang and M. Sarrafzadeh "Dragon2006: Blockage-Aware Congestion-Controlling Mixed-Size Placer," *ISPD*, pp. 209-211, 2006.

[131] K. Takahashi, K. Nakajima, M. Terai, and K. Sato, "Min-cut Placement with Global Objective Functions for Large Scale Sea-of-gates Arrays," *IEEE Trans. on CAD*, vol. 14, no. 4, pp. 434-446, 1995.

[132] X. Tang, R. Tian, M. D. F. Wong, "Optimal Redistribution of White Space for Wire Length Minimization," *ASP-DAC*, pp. 412-417, 2005.

[133] J. W. Tschanz et. al, "Adaptive Body Bias for Reducing Impacts of Die-to-Die and Within-Die Parameter Varations on Microprocessor Frequency and Leakage," *IEEE JSSC*, vol. 37, no. 11, pp. 1396-1402, 2002.

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

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

[136] N. Viswanathan, M. Pan and C. Chu, "FastPlace 2.0: An Efficient Analytical Placer for Mixed-Mode Designs," *ASP-DAC*, pp. 195-200, 2006.

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

[138] J. Vygen, "Algorithms for Large-Scale Flat Placement", *DAC*, pp. 746-751, 1997.

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

[140] E. Wein and J. Benkoski, " Hard macros will revolutionize SoC design," *EE Times*, August 20, 2004. http://www.eetimes.com/news/design/showArticle.jhtml?articleID=26807055

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

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

[143] X. Yang, B. K. Choi, and M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-die Standard-cell Placement," *IEEE Trans. on CAD*, vol. 22, no. 4, pp. 410-419, April 2003. (*ISPD* 2002, pp. 42-49).

[144] X. Yang, R. Kastner and M. Sarrafzadeh, "Congestion Estimation During Top-down Placement," *IEEE Trans. on CAD*, vol. 21, no. 1, pp. 72-80, 2002.

[145] M. C. Yildiz and P. H. Madden, "Improved Cut Sequences for Partitioning Based Placement," *DAC*, pp. 776-779, 2001.