Multiobjective Placement Optimization for High-performance Nanoscale Integrated Circuits

by

Myung Chul Kim

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
2012

Doctoral Committee:

Professor Igor L. Markov, Chair
Professor David T. Blaauw
Professor Mark S. Daskin
Professor Pinaki Mazumder

To my family and friends

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Multiobjective Placement Optimization
for High-performance Nanoscale Integrated Circuits

by
Myung Chul Kim

Chair: Igor L. Markov

With aggressive scaling of semiconductor manufacturing technology in recent decades, the complexity of integrated circuits has increased rapidly leading to multi-million gate chips that require over ten metal routing layers. At current and future technology nodes, semiconductor devices are connected by narrower and more resistive wires, shifting the performance bottleneck from gate delay to interconnect delay. These trends confound modern design technologies for timing closure and require major improvements in physical design automation to maintain the current pace of innovation in chip architecture.

Modern VLSI design flows require considerable effort and time in physical layout, where transistor locations affect nearly all downstream optimizations during timing closure. However, despite impressive improvements developed in academia and industry during the last decade, state-of-the-art algorithms for placement leave room for improvement both in quality and speed. Additionally, mainstream wirelength-driven placement algorithms are not geared for optimizing various objectives that are required by advanced VLSI processes and design styles.

Our research addresses new challenges in physical optimization by (i) identifying the necessary new objectives, constraints and concerns imposed by contemporary and future semiconductor technologies, (ii) integrating these objectives with the existing objectives and tools, and (iii) developing new computational techniques to enhance scalability and robustness. We present new algorithms and methodologies for placement optimization subject to various constraints. In particular, we develop a standalone wirelength-driven global placement algorithm to significantly improve quality of standard-cell locations and decrease runtime. This algorithmic framework was recently adopted in the industry and has been extended by several university groups to support multiobjective optimization. In addition, our research shows how to integrate routability analysis within placement optimization, which is becoming increasingly important at upcoming semiconductor technology nodes. Experimental results indicate that the produced placements are significantly easier to route. We further enhance wirelength-driven placement using a multilevel framework and novel combinatorial optimization techniques. To broaden the scope of placement optimization, we study the theoretical aspects of our placement algorithms, and develop a variety of extensions: to different interconnect models, macro placement, and timing-driven placement. Another such extension is a placement framework that significantly improves the handling of datapath designs.

PART I

Introduction & Background

CHAPTER I

Circuit Placement in Physical Design

With ever-increasing chip complexity and expansive global communication, the number of components on an integrated circuit (IC) has doubled over every 18 months, according to Moore's law. Traditionally, device delays have dominated chip performance. With current technology trends, however, the bottleneck lies with interconnect delay, as device delays improve faster than interconnect delay and the amount of interconnect grows superlinearly with the number of components (except for grid-like circuits such as memory blocks). These trends pose significant challenges to IC designers who must also optimize other performance metrics, such as power consumption and area, and close timing [58]. Therefore, during the VLSI optimization process, chip designs often pass through several iterations of *placement*, *routing*, and *timing optimization* transforms.

While every design step ultimately affects timing closure, circuit placement has been and still remains at the core of physical design as a gating factor for downstream optimizations [10]. This process of computing non-overlapping locations for all circuit components not only significantly impacts the final product's delay, power, and area, but also affects the success of subsequent design-flow steps.

The first algorithms for circuit placement were developed independently at Bell Labs and IBM Research in the 1960s following the divide-and-conquer paradigm [19]. These algorithms motivated heuristics for balanced graph-partitioning by Kernighan and Lin [83] that minimize edge cut and, later, by Fiduccia and Mattheyses [51]. In the mid 1980s, circuit placement was a key application of the newly invented simulated annealing methods [56, 74, 129] . However, after rigorous technology scaling, simulated annealing was no longer scalable to handle the 1990s industrial designs. As a result, there academic researchers focused on scalable placement algorithms, and the divide-and-conquer framework regained momentum when it was combined with bottom-up clustering and multilevel partitioning [5, 23, 121, 136, 137]. Linear programming [116] and network flows [48] were also tried with limited success.

In the mid 2000s, increasing transistor density again demanded faster algorithms with better performance. IBM Research spearheaded the ISPD 2005 and 2006 placement contests and inspired the creation of many academic placers. In particular, analytical placement techniques (e.g., [26, 79, 140]) have generated a great deal of attention due to their superior performances at the ISPD 2005 and 2006 placement contests [107]. This joint industry-academic initiative motivated further advances in placement, such as those that consider routability [57,64,85,98,133]. Recently, analytical placement has been integrated with more comprehensive optimizations that can reduce interconnect by restructuring the circuit [10], and explicitly consider timing optimization [24, 77, 142], TSV count minimization for 3-d IC designs [63], buffer planning [35, 128] or other design objectives [95].

1.1 Challenges

Not only circuit placement serves as the foundation of many other optimization steps, but it is also often invoked between design steps, especially after timing analysis, physical synthesis, and congestion improvement. Therefore, having a fast, scalable, and robust placement engine is crucial for faster turnaround and lower time-to-market. However, with the sheer number of components and the increased circuit complexity, finding non-overlapping locations for all circuit components while meeting every performance constraint is becoming increasingly difficult. To improve scalability, circuit placement is divided into three steps: *global placement*, where every component is assigned a "coarse" and possibly overlapping location, *legalization*, which shifts the global locations to avoid overlaps and fit rows and columns, as well as *detailed placement*, where component locations are swapped and slightly displaced to further optimize design metrics. Despite impressive improvements reported by researchers [107], global placement algorithms still suffer from several challenges: (i) quality and runtime tradeoff, (ii) integration with other optimizations, and (iii) lack of parallelism. Each of these challenges is outlined below and addressed in subsequent chapters.

Tradeoff between speed and solution quality. State-of-the-art algorithms for global placement can be categorized as (i) force-directed quadratic placers, and (ii) nonconvex optimization techniques. Force-directed quadratic placers model the total interconnect length by a quadratic cost function of component locations and minimize it by solving a large sparse system of linear equations. To discourage component overlap, *forces* are added pulling components away from high-density regions. On the other hand, placers

based on nonconvex optimization model interconnect length and also component overlap by more sophisticated differentiable functions. Then a combined cost function of interconnect length and overlap is constructed with placer-specific net weights (that may change dynamically) and is minimized by nonconvex optimization techniques. Minimizing the combined cost function provides more systematic and precise quality control of resulting placements than control afforded by force-directed methods. Not surprisingly, placers based on this method previously claimed the best results among academic implementations and industry software. However, placers based on nonconvex optimization are significantly slower than force-directed quadratic placers due to their complicated objective function and optimization algorithms, which is problematic for extremely large modern chip designs. A representative nonconvex-optimization placer mPL6 [26] outperforms a force-directed quadratic placer FastPlace3.0 [140] by more than 2% in solution quality, but is about 6.5 times slower.

Simplicity and integration with other optimizations. Traditional techniques for placement that optimize only total interconnect length are often found to be insufficient to handle diverse challenges arising from advanced VLSI process technologies. Placements highly optimized with respect to traditional metrics can lead to failures during wire routing. Producing high-quality placements is becoming increasingly difficult with dramatic increases in size and complexity of designs along with multiple objectives, such as ensuring routability, optimizing timing critical nets, aligning structured components, or limiting power density while minimizing the total interconnect length. Even though novel methods are needed to close timing and accomplish multiple design goals, such integration is highly

complex and challenging. The complexities in placement that prohibit such integration fall into three categories: (i) fast and conclusive evaluation of important objective function on intermediate placements, (ii) effective optimization of multiple objectives while limiting the total interconnect length increase, and (iii) an increase in the complexity of software.

Given that circuit components may overlap during global placement and may lack proper alignement (*illegal locations*), numerical estimates of target objectives on computed intermediate placements can be inconclusive and misleading. For instance, actual routes and interconnect delays are often hard to estimate when locations of components are not yet fully determined. Even with legal placements, accurate measurements of routability and timing can be intractable if full-fledged routers or sign-off quality timers are naively invoked between placement iterations. Reliable integration of multiple optimization requires stable placement algorithms, as well as (i) accurate and fast estimation techniques, (ii) strategic invocations of such estimation techniques, and (iii) effective methods to influence the intermediate placements. Additionally, an increase in the complexity of software due to the integration should be kept manageable, which may require redesign of algorithms. For example, clustered netlists in multilevel placers can obscure analysis of routing congestion and timing, and complicate circuit restructuring.

Support for parallelism. Effective parallelization of CAD optimizations often requires redesign and simplification of entire algorithms to use fewer components, especially standard solvers, to avoid well-known limits to parallelism described by Amdahl's law [13]. On the other hand, recent literature on parallel algorithms and GPGPU programming often focuses on algorithms that are easier to parallelize, but are not the fastest or best-

performing available [52, 71, 103]. Such results may be useful to illustrate specific parallelization techniques, but do not justify the need for parallelization. We believe that new EDA tool development should not solely focus on parallel processing, but rather on novel high-performance algorithms amenable to parallel processing.

As mentioned above, tools based on nonconvex optimization achieve the best results reported for academic implementations [36] and EDA vendor tools, but are significantly slower. To scale the basic nonconvex optimization framework, all tools in this family employ netlist clustering and multilevel extensions, sometimes at the cost of solution quality. Such multilevel placers perform many sequential steps, obstructing efficient parallelization, and therefore do not fully benefit from modern multicore CPUs.

1.2 Our Contributions

This dissertation includes the following contributions:

A simple and effective algorithm for global placement. Chapter III presents a self-contained, flat, force-directed placement algorithm for large-scale placement that outperforms prior art both in runtime and solution quality on standard benchmarks. The algorithm is iterative and maintains two placements — lower-bound and upper-bound placements that converge to a final solution. The upper-bound placement is produced by our new feasibility projection algorithm based on top-down partitioning and nonlinear scaling. Our implementation, SimPL, is self-contained and relies on a small set of optimization techniques, which provides a clear path to both instruction-level and thread-level parallelism. This framework was recently adopted in the industry (Chapter V) and has been extended by several university groups to support multiobjective optimization [14, 57, 152].

Chapter V extends the SimPL algorithm to further optimize interconnect length and handle movable macros by employing a new multilevel framework in conjunction with novel combinatorial optimization techniques. Our framework, called MAPLE (Multilevel Adaptive PLacEment for Mixed-size Designs), produces the best published results in total wirelength on a wide variety of publicly available benchmarks.

Routability-driven placement. In earlier technology generations, placement and routing algorithms were designed and implemented in separate software tools due to their complexity, even when the user interface exposed a single optimization to chip designers. After a placer generates row- and site-aligned, non-overlapping locations of components while minimizing interconnect length, a global router then routes all signal nets, with small total wirelength, subject to track capacity constraints. Yet, due to the increasing design complexity, solely minimizing traditional placement metrics no longer guarantees violation-free solutions after routing [12, 124]. Furthermore, highly-optimized placements may lead to routing congestion and failures when the placer is unaware of routing resource supply and demand. Chapter IV extends the SimPL placer to explicitly account for routability. We develop an accurate routing-congestion estimation technique to give the placer advance knowledge of congestion spots as well as the ability to respond early and often. These techniques are seamlessly incorporated in a simultaneous place-and-route framework for global placement, called SimPLR, as well as a congestion-aware detailed placement algorithm.

Part III introduces additional enhancements and extensions of our techniques.

Multiobjective placement optimization. Placement techniques driven entirely by interconnect length are often insufficient to address diverse algorithmic challenges, design specifications and constraints posed by advanced VLSI designs and technology nodes. Supporting a variety of such objectives early in placement has been highly desirable as it can accelerate physical design iterations and timing closure. Yet, such integration is sometimes considered prohibitive in the industry because it may sharply increase software complexity. Our research simplifies such integration by reducing the complexity of relevant algorithmic components and clarifying their roles in multiobjective optimization. Chapter VI provides theoretical justification for the SimPL algorithm and other placement algorithms derived from it along with an improved convergence analysis and control. To broaden the scope of placement optimization, it also proposes several extensions for handling various constraints imposed by new technologies and design styles. Chapter VII describes our joint work with IBM Research on integration of structure-awareness into placement with SAPT (Structure-Aware Placement Techniques). On *datapath* and *hybrid* designs, SAPT extracts relative orders between datapath groups and enforces alignment constraints. SAPT significantly improves overall quality of automatic placement on such designs, reducing the quality gap with manual placement.

1.3 Organization of the Dissertation

The rest of this dissertation is organized as follows: Part I outlines relevant background in VLSI modern physical design in general and placement algorithms in particular. Part II covers our work in VLSI placement: Chapters III and V describe our wirelength-driven

placement algorithms SimPL and MAPLE. Chapter IV focuses on routability-driven placement and introduces the SimPLR algorithm. Part III presents extensions and new techniques that broaden the scope of placement optimization. Chapter VI provides theoretical justification for our placement algorithms, generalizes them and develops several extensions. Chapter VII describes our approach to placing circuits with regular structures (*datapath*). Part IV concludes the dissertation and discusses open challenges in Chapter VIII.

CHAPTER II

State of the Art in Placement Algorithms

The success of placement determines all aspects of modern IC layout and physical synthesis [10] because it controls the amount of interconnect, which increasingly dominates on-chip resources and circuit performance [80]. Placement seeks the locations of circuit components on a fixed outline such that no overlap exists between pairs of modules and minimizes some cost metrics (e.g., interconnect length). A proven NP-hard combinatorial optimization problem, circuit placement is typically split into three steps: *global placement*, *legalization*, and *detailed placement*. (i) *Global* placement assigns every (movable) circuit module a coarse location such that the total interconnect length between all modules is minimized. (ii) *Legalization* removes all overlaps between modules and assigns them to rows and sites. A typical objective function of legalization is to minimize the total (linear or quadratic) displacements from the global placement solution. (iii) *Detailed* placement further optimizes the legalized placements, typically in an iterative manner by rearranging a small groups of modules in local regions. Among these three steps, global placement is generally considered the most critical due to its significant impact on the overall layout quality.

This chapter discusses foundational components of placement algorithms and prior state of the art in placement research. Section 2.1 presents essential concepts and building blocks of modern placement algorithms. Section 2.2 reviews leading wirelength-driven placement algorithms. Section 2.3 describes recent challenges in ensuring routability and prior art to address them. Section 2.4 discusses optimization of multiple placement objectives in modern VLSI designs.

2.1 Essential Concepts and Building Blocks

Circuit placement typically operates on a gate-level netlist, which consists of standard cells (NAND, NOR, MUX, half-adders, etc), macro modules, and interconnect. Each standard cell has rectangular footprint with well-defined area. Some standard cells drive multiple other cells — such interconnects are captured by signal nets. Given a netlist $\mathcal{N} = (E, V)$ with nets E and nodes (cells) V, *global placement* seeks node locations (x_i, y_i) such that the area of nodes within any rectangular region does not exceed the area of (cell sites in) that region.[1] Some locations of cells may be given initially and fixed. The interconnect objective optimized by global placement is the weighted Half-Perimeter WireLength (wHPWL). For node locations $\vec{\mathrm{x}} = \{x_i\}$ and $\vec{\mathrm{y}} = \{y_i\}$, wHPWL$_{\mathcal{N}}(\vec{x}, \vec{y})$= wHPWL$_{\mathcal{N}}(\vec{x})$+wHPWL$_{\mathcal{N}}(\vec{y})$, where

$$wHPWL_{\mathcal{N}}(\vec{x}) = \Sigma_{e \in E} w_e [\max_{i \in e} x_i - \min_{i \in e} x_i] \tag{2.1}$$

This piecewise-linear function lends itself to linear programming (LP) and min-cost max-flows, but these techniques have been successful only for smaller netlists. In large-scale

[1] In practice, this constraint is enforced for bins of a regular grid. The layout area is subdivided into equal, disjoint, small rectangles, and each rectangle limits total area of cells placed within.

placement, wHPWL$_{\mathcal{N}}$ is approximated by convex twice-differentiable functions $\Phi(\vec{x}, \vec{y})$ and optimized numerically by linear or nonlinear Conjugate Gradient as illustrated in quadratic approximations.

Constraints in placement include *legality*, *target utilization*, *routability*, *resource-type* constraints, etc.

$$(\vec{x}, \vec{y}) \in \mathcal{C}$$

which prohibit multiple pairs (x_i, y_i) from concentrating in small regions. The demands for physical on-chip resources (gate area or number of routes in a region) must not exceed available supplies/design constraint (area for placing logic gates, target utilization, number of routing tracks) [80]. This is typically expressed by inequalities, e.g., allowing at most $C_{j,k}$ placeable objects in grid-cell (j, k). These inequalities are easy to satisfy when no optimization is performed. Unlike Φ, the constraints are noncovex, as illustrated by constraints on locations of two non-overlapping rectangles. Another type of constraints — routability of modern IC layouts — is NP-hard to evaluate with sufficient accuracy [80]. Some layout regions may be blocked by fixed obstacles and unavailable to (x_i, y_i), leading to discrete choices, such as placing an object on one side of an obstacle. This inhibits smooth convex optimization and, historically, motivated specialized global-placement techniques tailored to variant objective functions and constraints [107].

Quadratic approximations are used in many placers. Consider a graph $\mathcal{G} = (E_{\mathcal{G}}, V)$ with edges $E_{\mathcal{G}}$, vertices V and edge weights $w_{ij} > 0$ for all edges $e_{ij} \in E_{\mathcal{G}}$. The *quadratic objective* $\Phi_{\mathcal{G}}$ is defined as

$$\Phi_{\mathcal{G}}(\vec{\mathrm{x}}, \vec{\mathrm{y}}) = \Sigma_{i,j} w_{i,j} [(x_i - x_j)^2 + (y_i - y_j)^2] \tag{2.2}$$

$\Phi_{\mathcal{G}}$ is separable and its x and y components are cast in matrix form [18, 134]

$$\Phi_{\mathcal{G}}(\vec{x},\vec{y}) = \frac{1}{2}\vec{x}^T Q_x \vec{x} + \vec{f}_x \vec{x} + \frac{1}{2}\vec{y}^T Q_y \vec{y} + \vec{f}_y \vec{y} + \text{const} \tag{2.3}$$

The Hessian matrices Q_x and Q_y that are derived from the netlist captures connections between pairs of movable objects and vectors $\vec{f}_x, \vec{f}_y$ reflect connections between movable and fixed objects. When sufficiently many nodes in a connected netlist are fixed,$\Phi_{\mathcal{G}}$ is strictly convex and can be optimized quickly [80, Section 4.3.2]. When Q_x is non-degenerate, $\Phi_{\mathcal{G}}(\vec{\mathrm{x}})$ is a strictly convex function with a unique minimum, which can be found by solving the system of linear equations $Q_x\vec{\mathrm{x}} = -\vec{\mathrm{c}}_x$. Solutions can be quickly approximated by iterative Krylov-subspace techniques, such as the Conjugate Gradient (CG) method and its variants [127]. Since Q_x is symmetric positive definite, CG iterations provably minimize the residual norm. The convergence is monotonic [138], but its rate depends on the spectral properties of Q_x, which can be enhanced by *preconditioning*. In other words, we solve the equivalent system $P^{-1}Q_x = -P^{-1}\vec{c}_x$ for a non-degenerate matrix P, such that P^{-1} is an easy-to-compute approximation of Q_x^{-1}. Given that Q_x is diagonally dominant, we chose P to be its diagonal, also known as the *Jacobi preconditioner*. Our placement algorithm (Section 3.3.3) deliberately enhances diagonal dominance in Q_x.

To approximate the HPWL by quadratic functions, one uses a *linearization* technique [132], adjusting the approximations at every global placement iteration. In particular, single-edge terms of the form $w_{ij}(x_i - x_j)^2$ are changed to $\frac{w_{ij}(x_i-x_j)^2}{|x'_i-x'_j|+\varepsilon}$ where the primed values are constants based on *the result of the last iteration* (a.k.a. the last *iterate*). Multipin nets are decomposed into sets of edges using stars, cliques or the Bound2Bound model [134].

The Bound2Bound net model [134]. To represent the HPWL objective by the quadratic objective, the netlist $\mathcal{N}$ is transformed in *two* graphs, $\mathcal{G}_x$ and $\mathcal{G}_y$, that preserve the node set V and represent each two-pin net by a single edge with weight $1/length$. Larger nets are decomposed depending on the relative placement of vertices — for each p-pin net, the *extreme* nodes (min and max) are connected to each other and to each *internal* node by edges, with the following weight

$$w_{x,ij}^{B2B} = \frac{1}{(p-1)|x_i - x_j|} \tag{2.4}$$

For example, 3-pin nets are decomposed into cliques with edge weight $1/2l$, where l is the length of a given edge. In general, this quadratic objective and the Bound2Bound (B2B) net decomposition capture the HPWL objective exactly, but only for the given placement. As locations change, the error may grow, necessitating multiple updates throughout the placement algorithm.

Most quadratic placers use the placement-independent star or clique decompositions, so as not to rebuild Q_x and Q_y many times [18, 140, 141]. Yet, the B2B model uses fewer edges than cliques ($p > 3$), avoids new variables used in stars, and is more accurate than both stars and cliques [134].

Quadratic placement example. Consider the graph $\mathcal{G}$ and edge weights w_{ij} in Figure 2.1. Quadratic placement minimizes the separable quadratic cost function $\Phi_{\mathcal{G}}$ in the x- and y- directions. For the x-direction,

$$\Phi_{\mathcal{G}}(\vec{\mathrm{x}}) = w_{12}(x_1 - x_2)^2 + w_{13}(x_1 - x_3)^2 + w_{1f_1}(x_1 - f1)^2$$
$$+w_{23}(x_2 - x_3)^2 + w_{34}(x_3 - x_4)^2 + w_{4f_2}(x_4 - f_2)^2$$

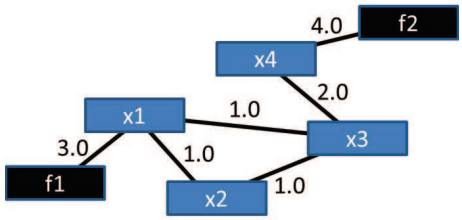


Figure 2.1: Blue boxes represent movable modules and black boxes represent fixed modules. Connections between modules are shown with black lines.

Setting the partial derivative to 0 (the condition for force equilibrium), we solve for the global minimum cost.

$$\frac{\partial \Phi_{\mathcal{G}}(\vec{\mathrm{x}})}{\partial x} = 0 \Leftrightarrow Q_x \vec{\mathrm{x}} = -\vec{\mathrm{c}}_x \Leftrightarrow \tag{2.5}$$

$$\begin{bmatrix} 5.0 & -1.0 & -1.0 & 0.0 \\ -1.0 & 2.0 & -1.0 & 0.0 \\ -1.0 & -1.0 & 4.0 & -2.0 \\ 0.0 & 0.0 & -2.0 & 6.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3.0f_1 \\ 0.0 \\ 0.0 \\ 4.0f_2 \end{bmatrix}$$

The connectivity matrix Q_x has entry w_{ij} in the ith row and jth column, and $-\vec{\mathrm{c}}_x$ has entry c_i in the ith row. The diagonal entries w_{ii} correspond to the sum of net weights of all connections to movable module i. The off-diagonal entries w_{ij} are calculated as the negative sum of net weights of connections between movable modules i and j, and the resulting connectivity matrix becomes symmetric. Each element c_x for a movable module i is calculated as the sum of $w_{ij} \cdot x_j$ where x_j is the pin location of each connected fixed module. With $(f_1, f_2) = (1.0,\ 3.5)$, a linear system solver finds a unique solution $\vec{\mathrm{x}} = [1.4762\ 1.9524\ 2.4286\ 3.1429]^T$ that minimizes the quadratic wirelength $\Phi_{\mathcal{G}}(\vec{\mathrm{x}})$.

2.2 Wirelength-driven Global Placement Algorithms

VLSI design flow often includes numerous stages of placement, routing and timing optimization transforms, and the quality of individual stages is ultimately evaluated by the timing closure of the design. Yet, wirelength is commonly used as a concrete goal in VLSI placement stage. This is because not only wirelength is easy to measure, but also it is closely related to the amount of overall routing resource demand and timing optimization effort. Given that significant portion of circuit delay is dominated by interconnect in modern high-performance VLSI designs, those with lower interconnect length can often achieve faster overall delay. Circuit placements with faster delay require smaller iterations of timing optimization transforms to close timing, enabling a shortcut in design time and effort. Therefore, state-of-the-art algorithms for placement commonly minimize wirelength that connects circuit modules while ensuring legality of module placement. Even in multiobjective placement, e.g., routability-driven or timing-driven placement, the choice-of the wirelength-driven global placement engine is paramount as demonstrated in the ISPD 2011 routability-driven placement contest [143].

State-of-the-art analytical algorithms for global placement form two families: (i) *force-directed* quadratic placers, such as Kraftwerk2 [134], FastPlace3 [140] and RQL [141], and (ii) *nonconvex optimization* techniques, such as APlace2 [79], NTUPlace3 [36] and mPL6 [26]. Force-directed quadratic algorithms model total net length by a quadratic function of cell locations and minimize it by solving a large sparse system of linear equations. To discourage cell overlap, *forces* are added pulling cells away from high-density areas. These forces are modeled by *pseudopins* and *pseudonets*, which extend the original

quadratic function [65]. They are updated after each linear-system solve until iterations converge. Nonconvex optimization models net length by more sophisticated differentiable functions with linear asymptotic behavior which are then minimized by advanced numerical analysis techniques [79]. Cell density is modeled by functional terms, which are more accurate than forces, but also require updates after each change to placement [36, 79]. Algorithms in both categories are used in the industry or closely resemble those in industry placers [78, 87, 141].

Tools based on nonconvex optimization achieve the best results reported for academic implementations [36] and EDA vendor tools, but are significantly slower, which is problematic for modern flat SoC placement instances with tens of millions of movable objects. To scale the basic nonconvex optimization framework, all tools in this family employ *netlist clustering* and *multilevel extensions*, sometimes at the cost of solution quality. Such multilevel placers perform many sequential steps, obstructing efficient parallelization. Moreover, clustering and refinement do not fully benefit from modern multicore CPUs. Due to their complexity, multilevel placers are also harder to maintain, improve, and combine with other physical-design techniques. In particular, clustered netlists complicate accurate static timing analysis, congestion maps and physical synthesis transformation, such as performance-driven buffering, gate sizing, fanin/fanout optimization, cloning, etc [10]. Hence, timing-closure flows often repeat global placement 3-4 times, alternating it with timing analysis, physical synthesis and congestion improvement.

State-of-the-art force-directed quadratic placers tend to run many times faster than nonconvex optimization, but also use multilevel extensions in their most competitive con-

figurations. Their solution quality is mixed. FastPlace3 underperforms mPL6, but the industry tool RQL closely related to FastPlace outperforms these two nonconvex placers. Kraftwerk2 is the only competitive *flat* placer (i.e., it does not use clustering) and rivals other force-directed quadratic placers in speed. However, it lags behind in solution quality and poses several challenges, such as quickly solving Poisson's equation, ensuring the convergence of iterations and avoiding halos of unused space around macros. Our experience indicates that the performance of Kraftwerk2 can be uneven, and stability can only be achieved with some loss of solution quality [82]. Several placers are described in the book [107] and journal papers [18, 36, 134].

2.3 Routability-driven Placement

In earlier technology generations, placement and routing algorithms were designed and implemented in separate software tools due to their complexity. After logic synthesis, large-scale placers generate row- and site-aligned, non-overlapping locations for cells with small interconnect length (HPWL). A global router then routes every signal net, seeking to minimize total wirelength, subject to track capacity constraints. However, at advanced technology nodes, the traditional placement metrics no longer suffice to quantify solution quality [12, 124] and consider congestion objectives. Furthermore, highly-optimized placements often lead to routing failures when a wirelength-driven global placer has no information about actual routes.

Traditional congestion-aware placement algorithms focused on allocating / injecting whitespace to congestive regions to improve routability [78, 98, 162]. However, due to the lack of interaction between the placement and the routing stages, routers may not honor the

resource allocation preserved by a placer. Previous work shows how to incorporate congestion analysis into global placement.

Congestion estimation methods [12]. A successful estimator, potentially based on a global router, must account for up to twelve metal layers with wire widths and spacings that differ by up to 20 times. Blockages and per-layer routing rules must be accounted as well. Other constraints include via spacing rules and limits on intra-gcell routing congestion. With the 2007 and 2008 ISPD contests [69, 108], several academic routers — NTHU-Route 2.0 [28], NTUgr [61], FastRoute 4.0 [159], BFG-R [67] — have emerged to account for some of these issues. More recent routers — PGRIP [158], PGR (SGR) [101], GLADE [29, 96] — have improved upon the solution quality and runtime of the routing solutions, and accounted for different layer directives.

Congestion maps. To estimate congestion, prior approaches can be divided into three main categories: (i) static congestion estimation, (ii) probabilistic congestion estimation, and (iii) global routing estimation. Traditionally, the first two options have been the most popular approaches, but due to the recent advances of global routers and the increased complexity of global routing, the last approach is gaining popularity. In addition, there is evidence that constructive methods, e.g., generating routes, is faster and more accurate than using probabilistic methods [156]. All three approaches are summarized in Table 2.1.

Congestion-driven placement. Once the congestion information is generated, the optimizations can be divided into four main approaches: (i) during global placement, (ii) external optimizations applied to intermediate solutions, (iii) during detailed placement, and (iv) external optimizations applied after placement. During global placement, the two

GENERAL APPROACH	SPECIFIC TECHNIQUES
STATIC	Rent's Rule [47, 49, 104, 163]
	net bounding box [37, 124]
	pin density [20, 166]
	counting nets in each region [148]
PROBABILISTIC	(uniform) wire density [133]
	pseudo-constructive wirelength [76]
	probabilistic pattern routing [155]
CONSTRUCTIVE	generating routes using A* on a collapsed (2-d) routing grid [156]
	using FastRoute [111] within an integrated framework [39]

Table 2.1: Prior congestion-driven placement techniques.

most popular optimizations are to either adjust the movable cells' location, or modify the cells themselves (e.g., cell bloating). These ideas must be adapted to specific placer types (e.g., quadratic, min-cut, annealing-based). Additional optimizations can be performed on intermediate placement solutions, During detailed placement, the most common approach is to swap cells so as to reduce congestion without significantly increasing wirelength. All four approaches are summarized in Table 2.2.

Generally, routability-driven placement has remained an open challenge. Until recentlyk, direct comparisons between different tools were difficult, partially due to the lack of modern large-scale standard benchmarks and common protocols. In the ISPD 2011 routability-driven placement contest, a new industry benchmark suite was released along with the official evaluation protocol [143]. Leading-edge academic tools submitted to the contest represented prior art and several emerging new techniques.

GENERAL APPROACH	SPECIFIC TECHNIQUES
DURING GLOBAL PLACEMENT	relocating the movable objects [73, 133, 140]
	cell bloating or cell inflation [20, 59]
	growing or shrinking placement regions [114]
INTERMEDIATE	local placement refinement [39]
DURING DETAILED PLACEMENT	linear placement based on Steiner length in small windows [72, 124]
	incorporating congestion into the objective function [165]
	cell swapping based on only congestion and overlap [39]
AFTER PLACEMENT	whitespace injection or reallocation [98, 124, 162]
	simulated annealing [37, 65, 145]
	linear programming [100]
	network flows [146, 147]
	shifting modules by expanding global routing grid tiles [165]
	using pin density and congestion map to spread and bloat cells [120]

Table 2.2: Prior congestion-driven placement techniques.

2.4 Optimizing Multiple Objectives in Placement

Next generation fabrication technologies and advanced VLSI processes raise new physical-design challenges, especially in placement. Traditional wirelength-driven placement algorithms are not geared for multiobjective optimization at new technology nodes and as a result, placement often requires a great deal of manual intervention to ensure timing closure. For example, placement engines that do not consider routability or cannot support macro placement ultimately require more design effort in subsequent stages. Several recent state-of-the-art placement algorithms seek to reconcile such diverse demands early in placement.

Large macro placement. To facilitate fast-growing design complexity, modern VLSI designs often integrate thousands of predesigned macros due to the massive reuse of Intellectual Property (IP) modules (microprocessors, controllers, signal/graphic processors) or embedded memories. Traditional sea-of-gates ICs design style with a handful of macros is being replaced by the sea-of-hard-macros design style [154]. In a traditional design flow, macro placement mostly occurs in floorplanning, and locations are fixed at the designers' discretion. Such flow typically does not achieve the optimal macro placement given that it cannot fully consider interconnect between macros and standard cells. With macros being responsible up to 50%-70% of the total area [154], the impact of suboptimal macro placement solutions is more pronounced and often cannot be compensated for even by state-of-the-art standard cell placement, resulting in the failure of timing closure. In turn, macro and standard cell co-placement (or *mixed-size placement*) is highly desirable to achieve placements with superior quality for modern designs with movable macros. However,

macros can differ greatly in aspect ratios, sizes and pin locations with standard cells, which causes challenges to traditional standard cell-based analytic placement algorithms. Existing approaches to mixed-size placement can be largely classified into three categories: (i) *two-stage* approach, (ii) *floorplanning* methods, and (iii) one-stage approach. In the two-stage approach, placers first determine legal locations of macros and performs standard cell placement while macros are fixed [32, 33]. In contrast to the traditional forward flow, the macro-placement stage often requires an initial placement for global positioning, typically obtained by using a traditional standard cell placer. A common objective of macro placement in the two-stage approach is to preserve the relative placement among macros acquired from initial placement while ensuring the legality. For example, [33] explicitly packs macros around chip / predefined boundaries to maximize routability of circuits. In the flooringplanning-based methods, placers determine locations of macros cells and standard cells constructively. Placers based on these methods [121, 160] partition the layout regions considering clustered block sizes and assign these clusters / macro cells to partitions and floorplanning is performed as necessary in the subregion. The membership and locations of macros are then incrementally refined over subsequent placement iterations. Even though these placers maintain overlap-free macro solutions during placement iterations, they often do not generate competitive solutions due to intrinsic limitations of the partitioning-based approach. In the one-stage approach, placers represent cell sizes with differential functions and perform simultaneous standard cell and macro placement while allowing macro overlaps during the analytic placement [26, 36, 79]. However, analytical placers sometimes cannot reliably generate legal placement especially in the presence of

big macros and/or high design utilization [33, 160]. Recent macro-placement algorithms try to further optimize wirelength by employing flipping and rotation moves for macros as demonstrated in [32, 33, 62, 160].

Timing-driven placement. While traditional placers are not explicitly timing-aware, they can satisfy timing constraints implicitly by wirelength minimization. Since interconnect delay is dominant in modern designs and is linearly proportional to interconnect length with proper buffering, scaled interconnect length is a good first order approximation of the overall delay. However, there exists a gap between the total wirelength and actual delay, and the wirelength of individual timing-critical paths. Since traditional wirelength-driven placers do not try to regulate the wirelengths among different paths and fail to consider gate delays, even highly wirelength-optimized solutions can have considerable timing violations on critical (or near-critical) paths, which are often hard to compensate for by timing transformations during later timing optimization stages, such as buffering and gate resizing.

To adress performance optimzation challenges, timing-driven placement algorithms have been studied extensively. Timing-driven placement typically assumes an initial wirelength-optimized placement (can be post-processed by rough buffering / fanout optimization) and timing analysis tools for the tentative solution to guide the timing-driven placement engine. The most critical aspect of timing-driven placement is the mechanism to use timing information to influence the core placement engines. Prior art can be classified as (i) *path-based* methods and (ii) *net-based* methods. Path-based methods [27, 70, 135] consider all or a subset of paths directly, and thus have more accurate timing view and control,

However, these methods have suffered from poor scalability due to the exponentially increasing number of paths [27]. On the other hand, net-based methods [77, 90, 117, 142] translate timing-analysis information into net weights or net constraints as a way to moderate computational complexity. The main idea of net-weighting is to assign heavier net weights to more timing-critical nets while minimizing the total weighted wirelength objective. However, these techniques often lack the global view of the full path, and may not be able to optimize timing effectively because they do not have exact control of net or path lengths. Recent research shows that hybrids of path-based and net-based approaches are promising [102].

Structured logic placement. As SoC frequencies exceed 1 GHz and shrinking design schedules necesitate design automation for microprocessor chips, the boundary between manually designed datapath logic and random logic macros is blurring. A new design style, referred to as *hybrid designs*, is emerging that integrates random logic with datapaths. Such datapaths often perform parallel operations on different bits of the same CPU word (often called the *bit-stack*), and their careful optimization is crucial for high frequency designs [109]. Prior work [68] has shown that, with separate placement engines, a dedicated datapath placer may overconstrain the random logic placement solution causing overall degradation in congestion and wirelength. A single placement flow handling both structures is extremely valuable, improving design time, quality, and saving development and maintenance costs. However, results in [149] demonstrate that even state-of-the-art placers are incapable of handling designs with regular structure, largely because they are unaware of datapath structure. Despite its importance, structure-aware placement is often

considered by researchers too difficult due to the complexity of the problem and the failure of numerous previous attempts.

Other technology-driven objectives. Methodologies to include other emerging technology trends are receiving considerable attention in the literature. Such objectives include power-driven clocked element distribution [95], power-network planning [97], TSV count minimization [35] and thermal-awareness [44] for heterogeneous / 3-d ICs, detailed routing (pin-density)-awareness [120], or manufacturability [34]. Research on multiobjective placement optimization seeks more custom-like quality of placement by extending to consider logic/hierarchy information [40] or even by restructuring [10, 30].

PART II

High-Quality Placement for Modern System-on-a-Chip Designs

CHAPTER III

SimPL: An Effective Placement Algorithm

We propose a self-contained, flat, quadratic global placement algorithm that is simpler than existing placers and easier to integrate into timing-closure flows. It maintains lower-bound and upper-bound placements that converge to a final solution. The upper-bound placement is produced by a novel *lookahead legalization* algorithm. The lower-bound placement is produced by solving a linear system with spreading forces. Our placer SimPL outperforms mPL6, FastPlace3, NTUPlace3, APlace2 and Capo simultaneously in runtime and solution quality, running 7.1 times faster than mPL6 (using a single thread) and reducing wirelength by 3% on the ISPD 2005 benchmark suite. Greater improvements are achieved on larger benchmarks. The new algorithm is amenable to parallelism. We report empirical studies with SSE2 instructions and up to eight parallel threads.

3.1 Introduction

Global placement currently remains at the core of physical design and is a gating factor for downstream optimizations during timing closure [10]. Despite impressive improvements reported by researchers [107] and industry software in the last five years, state-of-the-art algorithms and tools for placement suffer several key shortcomings which are

becoming more pronounced at recent technology nodes. These shortcomings fall into four categories: (i) speed, (ii) solution quality, (iii) simplicity and integration with other optimizations, (iv) support for multi-threaded execution. We propose the SimPL algorithm that simultaneously improves results in the first three categories and lends itself naturally to thread-level and instruction-level parallelism on multicore CPUs.

In this chapter, we develop a new, self-contained technique for global placement that ranks as a flat partition-based and force-directed placement algorithm. It maintains lower-bound and upper-bound placements that converge to a final solution. The upper-bound placement is produced by a novel *lookahead legalization* algorithm based on top-down geometric partitioning and nonlinear scaling. Our implementation outperforms published placers simultaneously in solution quality and speed on standard benchmarks. The lower-bound placement is produced by solving a linear system with spreading forces. Our algorithm is simpler, and our attempts to improve overall results using additional modules and extensions from existing placers (such as *netlist clustering* [26,79,140], *iterative local refinement* (ILR) [140], and median-improvement (*BoxPlace*) [82]) were unsuccessful.

In the remainder of this chapter, Section 3.2 introduces our key ideas and articulates our solution of the *force modulation* problem. The SimPL algorithm is presented in Section 3.3 along with complexity analysis. Extensions and improvements are discussed in Section 3.4, and empirical validation is described in Section 3.5. The use of parallelism is discussed in Section 3.6, and Section 3.7 summarizes our results.

3.2 Key Ideas

Analytic placement techniques first minimize a function of interconnect length, neglecting overlaps between standard cells, macros, etc. This initial step places many cells in densely populated regions, typically around the center of the layout. Cell locations are then gradually spread through a series of placement iterations, during which interconnect length slowly *increases*, converging to a final overlap-free placement (a small amount of overlap is often allowed and later resolved during detailed placement).

Our algorithm also starts with pure interconnect minimization, but its next step is unusual — most overlaps are removed using a fast *lookahead legalizer* based on top-down geometric partitioning and nonlinear scaling. Locations of movable objects in the legalized placement serve as *anchors* to coerce the initial locations into a configuration with less overlap, by adding pseudonets to baseline force-directed placement [65].

Each subsequent iteration of our algorithm produces (i) an almost-legal placement that *overestimates* the final result — through lookahead legalization, and (ii) an illegal placement that *underestimates* the final result — through linear system solver. The wirelength gap between lower-bound and upper-bound placements helps monitor convergence (Section 3.3.3).

Solving the force-modulation problem. A key innovation in SimPL is the interaction between the lower-bound and the upper-bound placements — it ensures convergence to a no-overlap solution while optimizing interconnect length. It solves two well-known challenges in analytic placement: (1) finding directions in which to spread the locations (*force orientation*), and (2) determining the appropriate amount of spreading (*force mod-*

ulation) [82, 141]. This is unlike previous work, where spreading directions are typically based on *local information*, e.g., placers based on nonconvex optimization use *gradient* information and require a large number of expensive iterations. Kraftwerk2 [134] orients spreading forces according to solutions of Poisson's equation, providing a global perspective and speeding up convergence. However, this approach does not solve the force-modulation problem, as articulated in [82].[1] The authors of RQL [141], which can be viewed as an improvement on FastPlace, revisit the force-modulation problem and address it by a somewhat *ad hoc* limit on the magnitude of spreading forces. In our work, the *lookahead legalization algorithm* (Section 3.3.2), invoked at each iteration, determines both the direction and the magnitude of spreading forces. It is global in nature, accounts for fixed obstacles, and preserves relative placement to ensure interconnect optimization and convergence. Our placement algorithm does not require exotic components, such as a Poisson-equation solver used by Kraftwerk; our C++ implementation is self-contained.

Global placement with lookahead. The legalized upper-bound placements constructed at every iteration of our placer can be viewed as *lookahead* because they are used only temporarily and not refined directly. They pull cell locations in lower-bound placements not just away from dense regions, but also toward the regions where space is available. Such *area lookahead* is particularly useful around fixed obstacles, where local information does not offer sufficient guidance. While not explored in this chapter, similar *congestion lookahead* and *timing lookahead* based on legalized placements can be used to integrate our placement algorithm into modern timing-closure flows.

[1] The work in [82] performs force modulation with *line search* but is not currently competitive with state of the art.

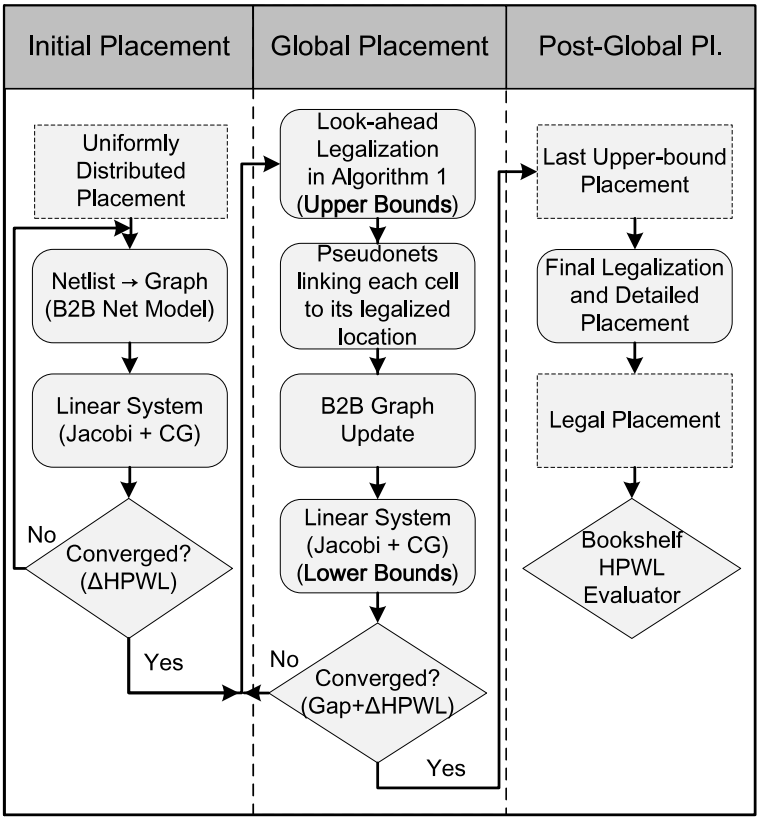


Figure 3.1: The SimPL algorithm uses placement-dependent B2B net model, updated on every iteration. *Gap* refers to the difference between upper and lower bounds.

3.3 The SimPL Global Placement Algorithm

Our placement technique consists of three phases: initial placement, global placement iterations and post-global placement (Figure 3.1). Initial placement, described next, is mostly an exercise in judicious application of known components. Our main innovation is in the global placement phase. Post-global placement is straightforward, given current state of the art.

3.3.1 Initial placement

Our initial-placement step is conceptually similar to those of other force-directed placers [134, 140, 141] — it entirely ignores cell areas and overlaps, so as to minimize a quadratic approximation of total interconnect length. We found that this step notably impacts the final result. Therefore, unlike FastPlace3 [140] and RQL [141], we use the more accurate Bound2Bound net model from [134] reviewed in Section 2.1. After the first quadratic solve, we rebuild the circuit graph because the B2B net model is placement-dependent. We then alternate quadratic solves and graph rebuilding until HPWL stops improving. In practice, this requires a small number of iterations (5-7), regardless of benchmark size, because the relative ordering of locations stabilizes quickly.

3.3.2 Lookahead legalization

Consider a set of cell locations with a significant amount of overlap as measured using bins of a regular grid. Lookahead legalization changes the global positioning of those locations, seeking to remove most of the overlap (with respect to the grid) while preserving the relative ordering. This task can be formulated at different geometric scales by varying

Algorithm 1 Lookahead Legalization by Top-down Geometric Partitioning and Nonlinear Scaling

Maximum allowed density γ, where $0 < \gamma < 1$
Current grid cell size
Floorplan with obstacles
Placement of cells
Queue of bin clusters $Q = \emptyset$

1: Identify γ-overfilled bins and cluster them // Fig. 3.2(a)
2: **foreach** cluster c **do**
3: Find a minimal rectangular region $R \supset c$ with density(R) $\leq \gamma$
4. $R.level$=1
5: Q.enqueue(R)
6: **while** !Q.empty() **do**
7: B=Q.dequeue()
8: **if** (Area(B) $<$ 4·grid cell size || $B.level \geq 10$) **then**
9: **continue**
10: M={movable cells in B}
11: **if** ($B.level$ % 2 == 0) **then** axis_direction D=HORIZ
12: **else** axis_direction D=VERT
13: C_c=D-aligned cutline to evenly split cell area in M
14: C_B=D-aligned cutline to evenly partition whitespace in B
15: (S_0, S_1)={two sub-regions of B created by cutline C_c}
16: M_0={movable cells in S_0}
17: M_1={movable cells in S_1}
18: (B_0, B_1)={two sub-regions of B created by cutline C_B}
19: Perform NONLINEAR SCALING on $M_0 \perp$ to D in B_0
20: Perform NONLINEAR SCALING on $M_1 \perp$ to D in B_1
21: $B_0.level$=$B_1.level$=$B.level$+1
22: Q.enqueue(B_0)
23: Q.enqueue(B_1)
24: **end while**
25: **end foreach**

the grid. The quality of lookahead legalization is measured by its impact on the entire placement flow. Our lookahead legalization is based on top-down recursive geometric partitioning and nonlinear scaling, as outlined in Algorithm 1. Cutlines C_c and C_B are chosen to be vertical at the top level and they alternate between horizontal and vertical directions with each successive level of top-down geometric partitioning.

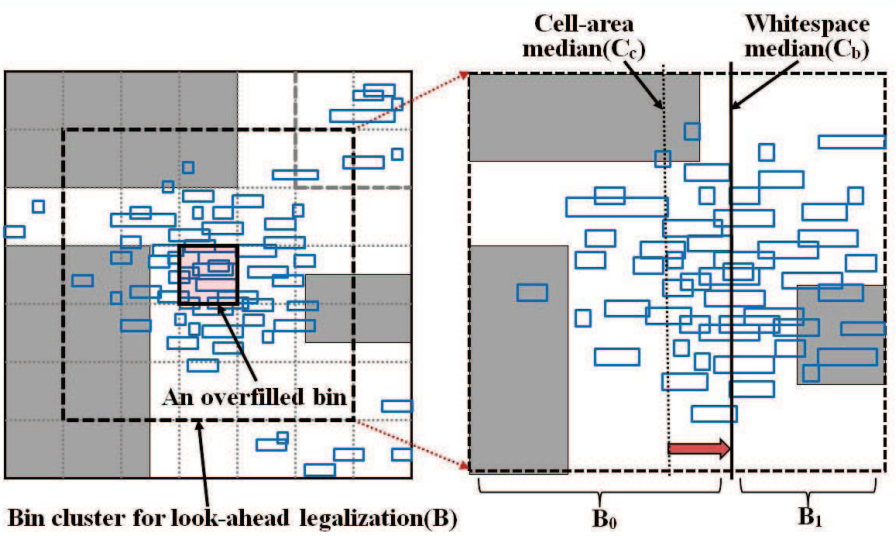


Figure 3.2: Clustering of overfilled bins in Algorithm 1 and adjustment of cell-area to whitespace median by nonlinear scaling (also see Figure 3.3). Movable cells are shown in blue, obstacles in solid gray.

Handling density constraints. For each grid bin of a given regular grid, we calculate the total area of contained cells A_c and the total available area of cell sites A_a. A bin is γ-*overfilled* if its *cell density* A_c/A_a exceeds given density limit $0 < \gamma < 1$. Adjacent γ-overfilled bins are clustered by Breadth-First Search (BFS), and lookahead legalization is performed on such clusters. For each cluster, we find a minimal containing rectangular region with density $\leq \gamma$ (these regions can also be referred to as *clusters*). A key insight is that overlap removal in a region, which is filled to capacity, is more straightforward because the absence of whitespace leaves less flexibility for interconnect optimization.[2] If relative placement must be preserved, overlap can be reduced by means of x- and y-sorting with subsequent greedy packing. The next step, *nonlinear scaling*, implements this intuition, but relies on cell-area cutline C_c chosen in Algorithm 1 and shifts it toward

[2]In the presence of whitespace, the placer can move cells around without changing their relative ordering [8]. Removing whitespace suppresses this degree of freedom, giving fewer choices to the placer.

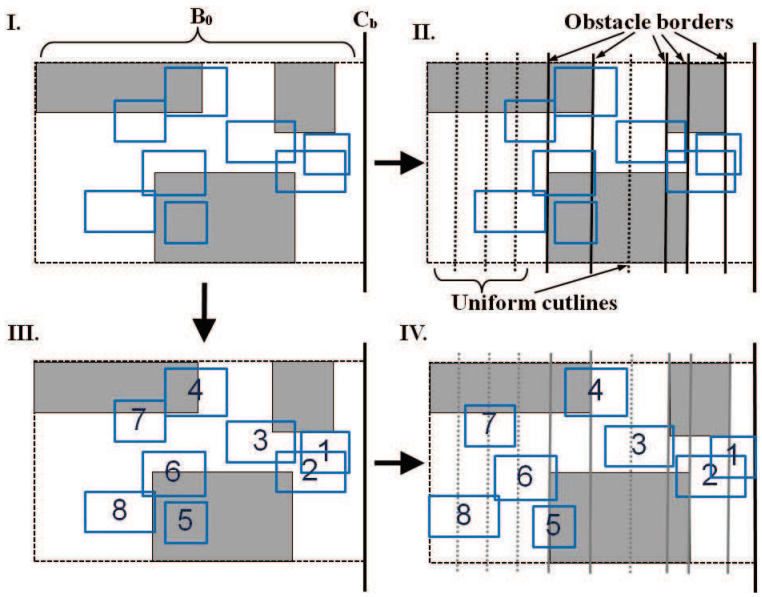


Figure 3.3: Nonlinear scaling in a region with obstacles (I): the formation of C_B-aligned stripes (II), cell sorting by distance from C_B (III), greedy cell positioning (IV).

the median of available area C_B in the region, so as to equalize densities in the two sub-regions (Figure 3.2).

Nonlinear scaling in one direction is illustrated in Figure 3.3, where a new region was created by a vertical cutline C_B during top-down geometric partitioning. This region is subdivided into vertical stripes parallel to C_B. First, cutlines are drawn along the boundaries of obstacles present in this region. Each vertical stripe created in this process is

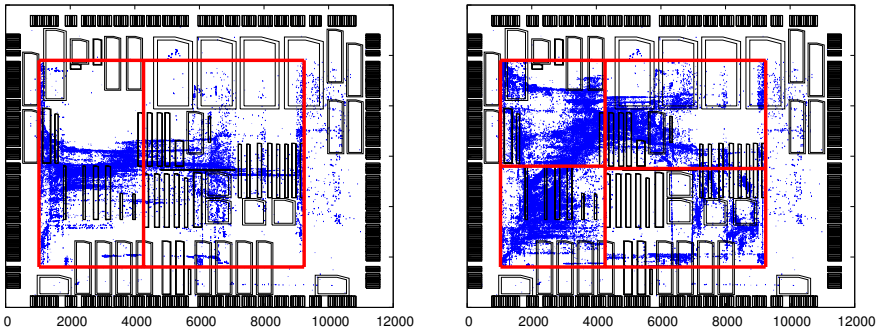


Figure 3.4: Nonlinear scaling after the first vertical cut and two horizontal cuts (ADAPTEC1) from intermediate steps between iterations 0 and 1 in Figure 3.7.

further subdivided (by up to 10 evenly distributed cutlines) if its available area exceeds $1/10$ of the region's available area. Movable cells in the corresponding sub-region created by C_c are then sorted by their distance from C_B and greedily packed into the stripes in that order. In other words, the cell furthest from the cutline is assigned to the furthest stripe. Each subsequent cell is assigned to the furthest stripe that is not filled yet. For each stripe, we calculate the available site area A_a and consider the stripe filled when the area of assigned cells reaches γA_a. Cell locations within each stripe are linearly scaled from current locations (non-linearity arises from different scaling in different stripes).

Lookahead legalization applies nonlinear scaling in alternating directions, as illustrated in Figure 3.4 on one of ISPD 2005 benchmarks. Here, a region R is selected that contains overfilled bins, but is wide enough to facilitate overlap removal. R is first partitioned by a vertical cutline, after which nonlinear scaling is applied in the two new sub-regions. Subsequently, lookahead legalization (Algorithm 1) considers each sub-region individually and selects different horizontal cutlines. Four rounds of nonlinear scaling follow, spreading cells over the region's expanse (Figure 3.4).

Despite a superficial similarity to cell-shifting in FastPlace [140], our nonlinear scaling does not use cell locations to define bins/ranges, or map ranges onto a uniform grid.

Cutline shifting. Median-based cutlines are neither necessary nor sufficient for good solution quality. We therefore adopt a fast cutline positioning technique from [119]. On benchmarks whose obstacles cover $<20\%$ of total sites area, we find cutline positions C_c minimizing net cut for the top two levels of top-down geometric partitioning, with $<60\%$ of cell area per partition. We record the ratio ρ of cell areas in the two partitions and adjust

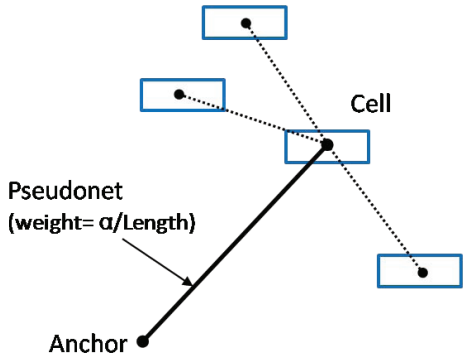


Figure 3.5: An anchor with a pseudonet.

the region's C_B cutline to the position that partitions the region's available area with the same ratio ρ. A related technique called ACG was developed at IBM in the context of min-cut placement, and their paper [8] describes relevant intuition.

3.3.3 Global placement iterations

Using legalized locations as anchors. Solving an unconstrained linear system results in a placement with significant amount of overlap. To pull cells away from their initial positions, we gradually perturb the linear system. As explained in Section 3.3.2, at each iteration of our global placement, top-down geometric partitioning and nonlinear scaling generates a roughly legalized solution. We use these legalized locations as fixed, zero-area anchors connected to their corresponding cells in the lower-bound placement with artificial two-pin pseudonets. Furthermore, following the discussion in Section 2.1, we note that connections to fixed locations do not increase the size of the Hessian matrix Q, and only contribute to its diagonal elements [80, Section 4.3.2]. This enhances diagonal dominance, condition number of $P^{-1}Q$, and the convergence rate of Jacobi-preconditioned CG.

In addition to weights given by the B2B net model on pseudonets, we control cell movement and iteration convergence by multiplying each pseudonet weight by an addi-

tional factor $\alpha > 0$ computed as $\alpha = 0.01 \cdot (1 + \text{iterationNumber})$. At early iterations, small α values weaken spreading forces, giving greater significance to interconnect and more freedom to the linear system solver. As the relative ordering of cells stabilizes, increasing α values boost the pull toward the anchors and accelerate the convergence of lower bounds and upper bounds.

Grid resizing. To identify γ-overfilled bins, we overlay a uniform grid over the entire layout. The grid size is initially set to $S_{init} = 100 \times 100$ to accelerate the lookahead legalization. However, in order to accurately capture the amount of overlap, the grid cell size decreases by $\beta = 1.06$ at each iteration of global placement until it reaches 4 times the average movable cell size.[3] Grid resizing also affects the clustering of γ-overfilled bins during lookahead legalization (Section 3.3.2), effectively limiting the amount of cell movement and encouraging convergence at later iterations. A progression of global placement is annotated with HPWL values in Figure 3.7. The upper-bound placements on the right appear blocky in the first iteration, but gradually refine with grid resizing.

Convergence criteria. A convergence criterion similar to that in Section 3.3.1 can be adopted in global placement. We alternate (1) lookahead legalization, (2) updates to anchors and the B2B net model, and (3) solution of the linear system, until HPWL of solutions generated by lookahead legalization stops improving. Unlike in the initial placement step, however, HPWL values of upper-bound solutions oscillate during the first 4-7 iterations, as illustrated in Figure 3.6. To prevent premature convergence, we monitor the gap between the lower and upper bounds. Global placement continues until (1) the gap is

[3]This is similar to mesh sizing for finite-element methods in numerical analysis and especially in adaptive mesh refinement. Parameters can be optimized for benchmark suites using binary search. However, we have not tuned parameters to individual benchmarks.

reduced to 25% of the gap in the 10th iteration and upper-bound solution stops improving or (2) the gap is smaller than 10% of the gap in the 10th iteration. On the ISPD 2005 benchmark suite, this convergence criterion entails 26-47 iterations of global placement. The final set of locations (global placement) is produced by the last lookahead legalization as shown in Figure 3.1.

Convergence is guaranteed by the increasing weights of pseudonets. At each iteration, these pseudonets pull the lower-bound placement toward a legalized upper-bound placement. As the lower-bound placement becomes closer to a legal placement, it exhibits a decreasing amount of cell overlap. This, in turn, results in smaller cell displacements during lookahead legalization. In the extreme, very high pseudonet weights force the lower-bound placement so close to the upper-bound placement, that lookahead legalization does not change it, resulting in immediate convergence.[4] In practice, pseudonet weights are changed gradually to ensure low interconnect length. After the first few iterations, one typically observes monotonic convergence, as illustrated in Figure 3.6.

3.3.4 Asymptotic complexity analysis

Modern placement algorithms are too complicated for asymptotic complexity analysis, but the bottlenecks of the SimPL algorithm yield to traditional analysis.

The runtime of global placement iterations is dominated by the Conjugate Gradient (CG) solver and lookahead legalization. The complexity of each CG invocation is $O(m\sqrt{\kappa})$, where κ is the conditioning number of the matrix and m is the number of nonzero elements [138]. The number of nonzeros reflects the number of graph edges

[4] This convergence argument only assumes that lookahead legalization does not change an upper-bound placement. It does not make any other assumptions about the lookahead legalization algorithm or consistency of its results between iterations. Neither does it say anything about the quality of results.

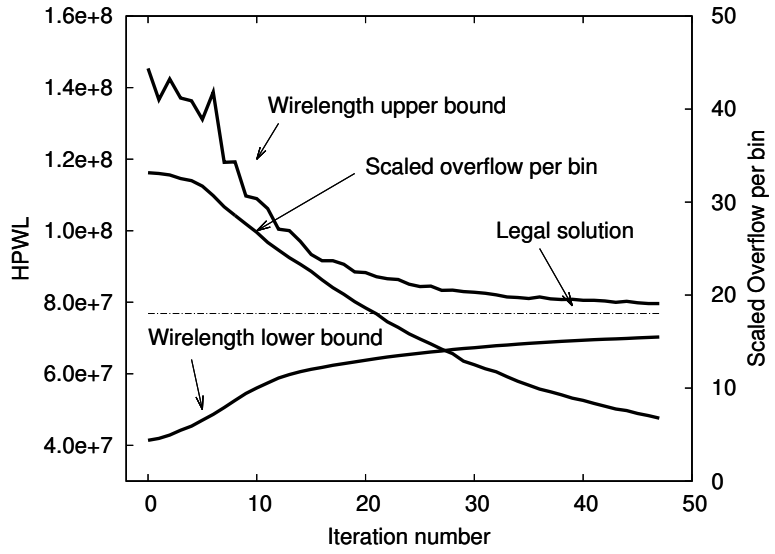


Figure 3.6: Lower and upper bounds for HPWL, the scaled overflow per bin of the lower-bound placement at each iteration, and HPWL of the legal placement (ADAPTEC1)

.

in the B2B model of the netlist. It grows linearly with the number of *pins* (cell-to-net connections) — a key size metric of a netlist. Another way to estimate the number of nonzeros is to observe that the average cell degree (the number of nets connected to a cell) is bounded by $d = 5$, or perhaps a slightly larger constant, for practical netlists.[5] Since $m \leq (d+1)n$ for n cells,[6] CG runs in $O(n\sqrt{\kappa})$ time.

Asymptotic runtime of lookahead legalization is dominated by sorting cell locations by their x and y coordinates because nonlinear scaling takes $O(n)$ time (several other linear-time steps take even less time in practice, therefore we do not discuss them). Given that lookahead legalization operates on blocks of progressively smaller size, we can separately consider its processing pass for the top-level blocks, then the pass for half-sized blocks, etc. Only $O(\log n)$ such passes are required for n cells. Each pass takes $O(n \log n)$ time because top-level blocks do not experience significant overlaps — in fact, each subsequent

[5]Even with large macros, whose number is limited by design area.

[6]Including diagonal matrix elements.

pass becomes faster because sorting is applied to smaller groups of cells. Hence, lookahead legalization runs in $O(n \log^2 n)$ time. We have observed that due to preconditioning, iteration counts in CG grow no faster than $\log n$, and each iteration takes linear time in n. Therefore one global placement iteration takes $O(n \log^2 n)$ time.

Empirically, SimPL requires <50 placement iterations, even for circuits with millions of cells. While the number of iterations might grow for larger circuits, this growth is very slow — possibly a polylog function of n. Empirical results in Section 3.5 show that SimPL's advantage in runtime and solution quality over its closest competitor (FastPlace3) increases on larger netlists. Min-cut placement (Capo) exhibits asymptotic complexity $O(n \log^2 n)$, but lags behind SimPL in runtime and quality.

Space complexity of our algorithms is linear in the size of the input, and our implementations require a modest amount of memory.

3.4 Extensions and Improvements

The algorithm in Section 3.3 can be improved in terms of runtime and solution quality. However, some of our attempts at improvement were unsuccessful. We report them here to warn the reader about their futility.

3.4.1 Selecting windows for lookahead legalization

During early global iterations, most movable cells of the lower-bound placement reside near the center of the layout region (Figure 3.7). In such cases, there is usually one expanded minimal rectangular region (cluster) that will encompass most of γ-overfilled bins. However, as global iterations progress, γ-overfilled bins will be scattered around the lay-

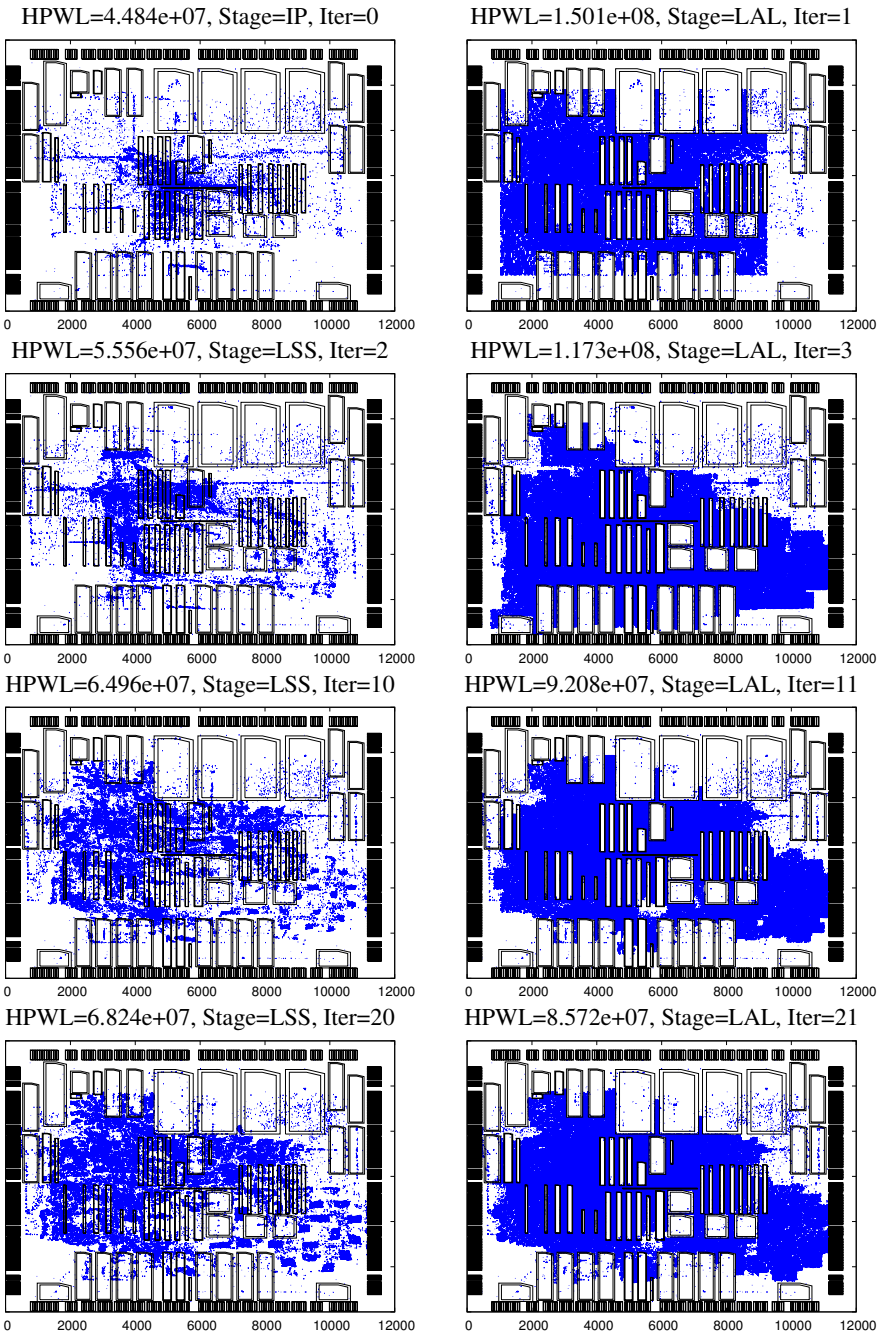


Figure 3.7: A progression of global placement snapshots from different iterations and algorithm steps (adaptec1). IP=Initial Placement, LAL=Lookahead Legalization, LSS=Linear System Solver. Left-side placements show lower bounds and right-side placements show upper bounds.

out region, and multiple clusters of bins may exist. In our implementation, we process γ-overfilled bins in the decreasing order of density. Each expansion stops when the cluster's density drops to γ or the cluster abuts the boundaries of previously processed clusters. This strategy may generate incompletely expanded clusters, especially in mid-stages of global placement iterations. However, as the densest bins are processed first, the number of regions with peak density is guaranteed to decrease at every iteration except when the peak density itself decreases. At each iteration of global placement, lookahead legalization is repeated up to ten times with increasing grid cell sizes until maximal density is decreased below γ.

3.4.2 Improving asymptotic complexity of lookahead legalization

As explained in Section 3.3.4, asymptotic runtime of lookahead legalization is largely determined by sorting cells by positions in directions perpendicular to cutlines. This sorting occurs at each level of top-down geometric partitioning. One way to improve asymptotic runtime complexity of lookahead legalization is to invoke sorting less often, given that lookahead legalization is to preserve the relative ordering among cells. Instead of sorting cells in each sub-region, we first establish two cell arrays sorted by x-coordinates and y-coordinates, respectively. At the second level of top-down geometric partitioning, two sub-regions inherit corresponding cells in-order from two sorted arrays of cells. In this way, if sorting is performed once at the top level of geometric partitioning, sorting at all successive levels can be replaced by selecting appropriate cells belong to current region in-order from two sorted arrays of higher level. This improves asymptotic runtime complexity of lookahead legalization from $O(n \log^2 n)$ time to $O(n \log n)$.

Benchmark size (#cells)	APLACE2.0		CAPO10.5		FASTPLACE3.0		MPL6		NTUPLACE3		SIMPL	
	HPWL	Time	HPWL	Time	HPWL	Time	HPWL	Time	HPWL	Time	HPWL	Time
AD1 211K	78.35	35.02	88.14	25.95	78.16	2.50	77.93	18.36	81.82	8.20	**76.87**	**2.47**
AD2 255K	95.70	50.57	100.25	36.06	93.56	3.66	92.04	19.91	**88.79**	7.57	90.37	**3.40**
AD3 452K	218.52	119.53	276.80	78.19	213.85	8.48	214.16	58.92	214.83	15.62	**206.38**	**6.68**
AD4 496K	209.28	131.57	231.30	79.32	198.17	7.10	193.89	55.95	195.93	16.18	**186.00**	**5.88**
BB1 278K	100.02	44.91	110.92	41.78	96.32	3.77	96.80	22.82	98.41	13.22	**95.85**	**3.47**
BB2 558K	153.75	100.96	162.81	80.55	154.91	9.62	152.34	61.55	151.55	26.17	**143.56**	**7.58**
BB3 1.10M	411.59	209.24	405.40	182.94	365.59	21.59	344.10	85.23	360.07	51.08	**336.19**	**13.02**
BB4 2.18M	871.29	489.05	1016.19	567.15	834.19	40.93	829.44	189.83	866.43	115.06	**796.78**	**37.37**
Geomean	**1.09**×	**15.34**×	**1.20**×	**12.17**×	**1.05**×	**1.20**×	**1.03**×	**7.10**×	**1.05**×	**3.05**×	**1.00**×	**1.00**×

Table 3.1: Legal HPWL (×10e6) and total runtime (minutes) comparison on the ISPD 2005 benchmark suite. Each placer ran as a single thread on a 3.2GHz Linux workstation. HPWL was computed by the GSRC Bookshelf Evaluator [1]. In benchmark names, AD stands for ADAPTEC and BB for BIGBLUE.

3.4.3 Unsuccessful attempts at improvement

Compared to other placement algorithms, SimPL uses a very modest set of interconnect optimizations. Therefore, we experimented with adding to SimPL several algorithms that were reported essential to the performance of other placers.

1. Our first attempt was to use netlist clustering to extend SimPL into a multilevel algorithm [26, 79, 140]. To this end, we implemented BestChoice clustering [106] used in FastPlace3 and were able to match its performance observed in FastPlace3 logs. This accelerated the initial CG solve in SimPL by about 2 times, with essentially the same quality of results, but unclustering increased the amount of cell overlaps, and the refinement techniques that we tried were either ineffective or too time-consuming.

2. In our second attempt, we implemented *iterative local refinement* (ILR) [140], which is a stage of FastPlace-global where it spends 40-50% of its runtime. ILR is a simple move-based algorithm that post-processes results of quadratic placement by relocating cells to nearby grid bins, while keeping track of both HPWL and cell density. ILR did improve the results of our early prototypes, but adding it to SimPL does not improve

final results. We believe that our *lookahead legalization* algorithm provides sufficient density control with a moderate increase in HPWL. We also tried, unsuccessfully, the median-improvement (*BoxPlace*) algorithm from [82], which moves single cells to their HPWL-optimal locations, while considering adjacent cells fixed.

3. In a third attempt, we evaluated *ad hoc force modulation* used in RQL [141] that neglects 10% strongest forces. Sweeping the range from 1% to 10% did not reveal any improvement in our experiments.

4. In our fourth attempt at improvement, we reordered vertices in the netlist to improve memory locality for each invocation of CG. This technique is often applied to the matrices of linear systems and is known to reduce cache misses and runtime. We implemented the Reverse Cuthill-McKee (RCM) reordering, which is standard in numerical analysis. The locality of nets has significantly improved. However, CG did not run faster on any of our benchmarks — the default ordering in our benchmarks was already good enough.

In summary, we obtain state-of-the-art results without extensions reported essential to other placers (FastPlace3 [140], FDP [82], and RQL [141]). We have also experimented with several preconditioners for CG, but found the simplest of them — the diagonal (Jacobi) preconditioner — to work best in our application.

3.5 Empirical Validation

Our implementation was written in C++ and compiled with g++ 4.4.0. Unless indicated otherwise, benchmark runs were performed on an Intel Core i7 Quad CPU Q660 Linux workstation running at 3.2GHz, using only one CPU core. We compared SimPL to other academic placers on the ISPD 2005 placement contest benchmark suite with target

density γ=1.0. Focusing on global placement, we delegate final legalization (into rows and sites) and detailed placement to FastPlace-DP [112], but post-process it by a greedy cell-flipping algorithm from Capo [23]. HPWL of solutions produced by each placer is computed by the GSRC Bookshelf Evaluator [1].

3.5.1 Analysis of our implementation

The SimPL global placer is a stand-alone tool that includes I/O, initial placement and global placement iterations. Living up to its name, it consists of fewer than 5,000 lines of C++ code and relies only on standard C++ libraries. There are four command-line parameters that affect performance — two for grid resizing (initial and step), and two for pseudonet weighting (initial and step). In all experiments we used default values described in Section 3.3.

Running in a single thread, SimPL completes the entire ISPD 2005 benchmark suite in 1 hour 18 minutes, placing the largest benchmark, BIGBLUE4 (2.18M cells), in 38 minutes using 2.1GB of memory. We report the runtime breakdown on BIGBLUE4 according to Figure 3.1, excluding 1.4% runtime for I/O.

Initial placement takes 5.0% of total runtime, of which 3.7% is spent in CG, and 1.3% in building B2B net models and sparse matrices for CG. **Global placement iterations** take 47.4%, of which 19% is in the CG solver, and 9.9% is in sparse matrix construction and B2B net modeling. Inserting pseudonets takes 0.8%, and lookahead legalization 17.7%. **Post-global placement** takes 46.2%, predominantly in detailed placement. Greedy orientation improvement and HPWL evaluation were almost instantaneous.

3.5.2 Comparisons to state-of-the-art placers

We compared SimPL to other placers whose binaries are available to us. We run each available placer,[7] including SimPL, in *default mode* and show results in Table 3.1. The HPWL results reported by APlace2 [79], Capo10.5 [23], [121] and mPL6 [26] were confirmed by the GSRC Bookshelf Evaluator. However, FastPlace3 [140] reported lower HPWL by 0.25% to 0.96%. For consistency, we report the readings of the GSRC Bookshelf evaluator.

SimPL found placements with the lowest HPWL for seven out of eight circuits in the ISPD 2005 benchmark suite (no parameter tuning to specific benchmarks was employed). On average, SimPL obtains wirelength improvement of 7.73%, 16.47%, 4.38%, 2.98%, and 4.48% versus APlace2, Capo10.5, FastPlace3, mPL6, and NTUPlace3 respectively. SimPL was also the fastest among the placers on all eight circuits. It is 7.01 times faster than mPL6, which appears to be the strongest pre-existing placer. SimPL is 1.20 times faster than FastPlace3, which has been the fastest academic placer so far.

While we managed to obtain almost all best-performing academic placers in binaries, RQL reportedly outperforms mPL6 in HPWL by a small amount [141]. Comparing our HWPL results to numbers in [141], we observe five wins for SimPL and three losses. RQL is 3.1 times faster than mPL6, making it more than twice as slow as SimPL.

3.5.3 Scalability study

To demonstrate SimPL's scalability to larger netlists, we generated variants of ISPD 2005 benchmarks with netlists that are twice as big with the same area utilization. In such

[7] The KraftWerk2 binary we obtained did not run on our system.

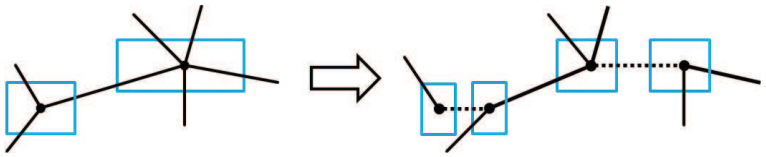


Figure 3.8: Generation of double-sized netlists.

a double-sized benchmark, each movable cell is split in two cells of smaller size, and each connection to the original cell is inherited by one of the split cells. Additionally, the two split cells are connected by a new two-pin net (Figure 3.8).

We compared SimPL to FastPlace3, mPL6, and NTUPlace3 on the double-sized benchmark suite and show results in Table 3.2. mPL6 could not finish bigblue4. For bigblue3, FastPlace-DP was unable to completely legalize solutions produced by FastPlace3-global, hence we post-processed FastPlace-DP with Capo10.5's legalizer.

SimPL was the fastest among the placers on all eight circuits. It is 8.96 times faster than mPL6, and 1.49 times faster than FastPlace3. SimPL also found placements with the lowest HPWL for six out of eight circuits in the double-sized ISPD 2005 benchmark suite (no parameter tuning to specific benchmarks was employed). Comparing results in Table 3.2 to those in Table 3.1, we observe that our placer has greater advantage on larger benchmarks. Furthermore, our runtime comparisons include detailed placement, but if SimPL is compared to FastPlace3-global without detailed placement, the average speed-up increases to 1.82 times from 1.58 times.

Compared to other placers, our implemantation uses a modest amount of memory — 1.65 times and 2.39 times less than mPL6 and NTUPlace3 rpdfectively, and 1.61 times more than FastPlace3. SimPL is using more memory than FastPlace3 when it constructs sparse matrices based on the Bound2Bound net model.

Ckts	FastPlace3.0			mPL6			NTUplace3			SimPL		
	HPWL	Time	Memory	HPWL	Time	Memory	HPWL	Time	Memory	HPWL	Time	Memory
ad1x2	80.30	3.67	**0.20**	79.11	24.95	0.43	80.20	19.6	0.72	**77.02**	**3.05**	0.36
ad2x2	98.88	6.06	**0.23**	93.64	38.00	0.82	**91.56**	17.6	0.86	92.41	**4.11**	0.36
ad3x2	258.71	13.47	**0.42**	232.87	88.71	1.11	225.32	44.7	1.62	**215.56**	**7.58**	0.71
ad4x2	219.35	11.54	**0.44**	206.24	85.72	1.16	197.90	39.6	1.76	**193.18**	**7.25**	0.69
bb1x2	97.93	5.68	**0.25**	100.37	30.35	0.53	99.33	22.0	0.93	**96.39**	**4.71**	0.43
bb2x2	164.74	12.13	**0.49**	159.24	79.84	1.22	154.47	44.2	1.94	**148.43**	**9.02**	0.77
bb3x2	515.61	49.89	**0.93**	395.26	172.96	3.38	**386.65**	154.9	3.82	403.40	**22.24**	1.44
bb4x2	865.30	56.36	**1.94**	fail	fail	fail	866.78	267.9	7.86	**854.64**	**42.43**	2.91
GM	**1.10**×	**1.49**×	**0.62**×	**1.04**×	**8.96**×	**1.65**×	**1.02**×	**4.40**×	**2.39**×	**1.00**×	**1.00**×	**1.00**×

Table 3.2: Legal HPWL (×10e6), total runtime (minutes), and peak memory usage (Gigabytes) comparison on the double-sized ISPD 2005 benchmark suite. The failure of mPL6 in bigblue4x2 is not caused by out-of-memory conditions.

3.6 Speeding up Placement Using Parallelism

Further speed-up is possible for SimPL on workstations with multicore CPUs.

3.6.1 Algorithmic details

Runtime bottlenecks in the sequential variant of the SimPL algorithm (Section 3.5.1) — updates to the B2B net model and the CG solver — can be parallelized. Given that the B2B net model is separable, we process the x and y cases in parallel. When more than two cores are available, we split the nets of the netlist into equal groups that can be processed by multiple threads. To parallelize the CG solver, we applied a coarse-grain *row partitioning* [60] scheme to the Hessian Matrix Q, where different blocks of rows are assigned to different threads using OpenMP [46]. A critical kernel operation in CG is the Sparse Matrix-Vector multiply (SpMxV). Memory bandwidth is a known performance bottleneck in a uniprocessor environment [55], and its impact is likely to aggravate when multiple cores access the main memory through a common bus. We reduce memory bandwidth demand of SpMxV by using the *CSR* (Compressed Sparse Row) [127] memory layout for the Hessian matrix Q.

In addition to thread-level parallelism, our implementation makes use of streaming SIMD extensions level 2 (SSE2) [115] (through g++ intrinsics) that perform several floating-point operations at once. SSE2 instructions are extensively used in our CG solver. Since SSE2 instructions are available in most modern CPUs, we used them in the default mode evaluated in Table 3.1 and Table 3.2. The overall speed-up due to parallelism varies between different hardware systems, as it depends on the relation between CPU speed and memory bandwidth.

```
// inner product of two float vectors x and y
float inner_product(vector<float>&x, vector<float>&y)
{
  float p_acc[4], inner_product=(float)0.;
  __m128 X, Y, acc = _mm_set_zero_ps();
  unsigned i;
  #pragma omp parallel for schedule(static)
  private(X,Y) lastprivate(i) reduction(+:acc)
  num_threads(NUM_CORES)
  for (i=0 ; i<=x.size()-4 ; i+=4)
  {
     X = _mm_load_ps(&x[i]);
     Y = _mm_load_ps(&y[i]);
     acc = _mm_add_ps(acc, _mm_mul_ps(X, Y));
  }
  _mm_store_ps(p_acc, acc);
  inner_product = p_acc[0]+p_acc[1]+p_acc[2]+p_acc[3];
  for ( ; i<x.size() ; i++)
        inner_product+=x[i]*y[i];
  return inner_product;
}
```

Listing 1. Sample code for OpenMP and SSE2 parallelization for inner-product operations.

After we parallelized the main bottlenecks, we noticed that lookahead legalization started consuming a significant fraction of overall runtime. Fortunately, top-down geometric partitioning and nonlinear scaling are amenable to parallelization as well. Notably, top-down partitioning generates an increasing number of sub-tasks of similar sizes which can be solved in parallel. Let Q be the global queue of bin clusters, as defined in Algorithm 1, and each thread has a private queue of bin clusters Q_i. First, we statically assign initial

Ckts	1 CORE	2 THREADS				4 THREADS				8 THREADS			
	C+S	CG	C+S	B2B	T&N	CG	C+S	B2B	T&N	CG	C+S	B2B	T&N
AD1	1.37	1.88	2.21	1.45	1.41	2.03	2.87	1.64	1.59	1.92	3.28	1.91	1.48
AD2	1.61	1.77	2.09	1.50	1.53	2.12	3.01	2.04	2.17	2.06	3.22	1.98	1.40
AD3	1.57	1.76	2.20	1.48	1.62	1.88	3.17	1.62	2.25	2.00	3.55	1.79	1.66
AD4	1.50	1.65	2.07	1.51	1.56	1.81	3.03	1.58	2.17	1.77	3.33	1.72	1.36
BB1	1.57	2.03	2.11	1.27	1.71	2.02	3.14	1.66	2.93	2.05	3.70	1.78	2.93
BB2	1.62	2.07	2.24	1.48	1.49	1.72	2.97	1.68	1.89	1.79	3.50	1.74	1.58
BB3	1.54	1.53	2.25	1.60	1.32	1.68	3.04	1.64	2.04	1.81	3.30	1.85	1.32
BB4	2.01	2.63	3.04	2.01	1.59	2.71	4.48	2.02	2.12	2.76	5.12	2.18	1.68
GM	**1.59**×	**1.89**×	**2.26**×	**1.53**×	**1.52**×	**1.98**×	**3.18**×	**1.73**×	**2.12**×	**2.00**×	**3.59**×	**1.86**×	**1.62**×

Table 3.3: Speed-up ratios for Conjugate Gradient (CG), B2B net model construction (B2B), and top-down geometric partitioning and nonlinear scaling (T&N) on the ISPD 2005 benchmark suite. Runtimes are compared to single-threaded execution without support of SSE instructions. C+S indicates our CG implementation with SSE instructions.

bin clusters to available threads such that each thread has similar number of bin clusters to start. After each level of top-down geometric partitioning and nonlinear scaling on such bin cluster, each thread generates two sub-clusters with similar numbers of cells. Then the thread t_i adds only one of two sub-clusters to its own cluster queue Q_i for the next level of top-down geometric partitioning and nonlinear scaling, while the remainder is added to the global cluster queue Q. Whenever Q_i of a thread t_i becomes empty, the thread t_i dynamically retrieves clusters from the global cluster queue Q. The number of clusters to be retrieved N is given by

$$N = \max(Q.size()/N_{threads}, 1)$$

where $N_{threads}$ is the total number of threads.

3.6.2 Empirical studies

As part of our empirical validation, we ran SimPL on an 8-core AMD-based system with four dual-core CPUs and 16GByte RAM. Each CPU was Opteron 880 processor running at 2.4GHz with 1024KB cache. Single-thread execution was compared to eight-thread execution as shown in Table 3.3. Our combination of multi-threading and SIMD

instruction-level parallelization was 1.6 times faster on average than parallelization based on multi-threading alone. Theoretically, using SIMD instruction-level parallelization may speed-up CG by at most four times. However, SIMD-based implementation of SpMxV only provided marginal speed-ups and was not worth the development effort. This is because irregular memory access patterns of SpMxV prohibit the aligned loading of values (*MOVAPS* or *_mm_load_ps* in Listing 1) to SSE registers. Nevertheless, SSE instructions were helpful in other parts of the code and contributed to the overall speed-up in global placement, as illustrated in Table 3.4.

We note that lookahead legalization operates on large datasets, but performs little computation per datum, which limits its performance by memory bandwidth. The amount of work per thread is so small that the overhead of thread creation outweighed the benefits. As a result, this part of SimPL scales poorly to >4 threads on available hardware, although this is probably not a fundamental limitation of the algorithm.

The overall speed-ups in global placement runtimes are shown in Table 3.4. Solution quality did not appreciably change, but peak memory usage increased by 1.91 times whereas runtime of global placement iterations was reduced by 2.4 times on average. The speed-ups saturate for more than 4 threads as lookahead legalization scales poorly. The initial placement stage was accelerated by about 3 times. While CG remained the runtime bottle neck of SimPL on 8 threads (36% of global placement), lookahead legalization became a close second ($> 31\%$ of global placement).

Ckts	2 THREADS		4 THREADS		8 THREADS	
	no SSE	SSE	no SSE	SSE	no SSE	SSE
AD1	1.70	1.71	1.76	2.03	1.71	2.23
AD2	1.75	1.73	1.91	2.43	1.90	2.35
AD3	1.59	1.72	1.79	2.30	1.81	2.40
AD4	1.55	1.65	1.75	2.24	1.67	2.26
BB1	1.75	1.67	2.17	2.56	2.18	2.67
BB2	1.70	1.72	1.67	2.22	1.66	2.37
BB3	1.49	1.75	1.71	2.28	1.65	2.28
BB4	1.94	2.12	2.01	2.55	2.03	2.69
Geomean	**1.68**×	**1.75**×	**1.84**×	**2.32**×	**1.82**×	**2.40**×

Table 3.4: Speed-up ratios for global placement on the ISPD 2005 benchmark suite. Runtimes are compared to single-threaded execution without support of SSE instructions.

3.7 Summary and Conclusions

In this chapter, we developed a new, flat, partition-based and force-directed quadratic global placer. Unlike other state-of-the-art placers, it is rather simple, and our self-contained implementation includes fewer than 5,000 lines of C++ code. The algorithm is iterative and maintains two placements — one computes a lower bound and one computes an upper bound on final wirelength. These two placements interact, ensuring stability and fast convergence of the algorithm. The upper-bound placement is produced by a new *lookahead legalization* algorithm, based on top-down geometric partitioning and nonlinear scaling, and converges to final cell locations. In contrast, all analytic algorithms we reviewed (both force-directed quadratic and nonconvex) derive their final solution from a lower-bound placement.

The use of partition-based techniques in upper-bound placements offers a solution to the force-modulation problem [82, 141] and removes the need for the so-called *hold forces* used by several force-directed placers.[8] As discussed in Section 3.2, upper-bound place-

[8]Hold forces are used to ensure that the current placement is a force equillibrium. Then move forces are added so that the placement can be improved. While this techniques is needed to ensure convergence of

ments perform an *area lookahead*[9] that is instrumental in the handling of layout obstacles. APlace2, NTUPlace3, mPL6, as well as some force-directed placers, model obstacles by additional smoothened *penalty* terms in the objective function. Not only such terms introduce extra work, but they also add imprecisions to modeling. For similar reasons, SimPL avoids netlist clustering used by other placers. We have implemented several other techniques essential to well-known placers, such as BoxPlace [82], ILR [140], and *ad hoc* force modulation [141], but they did not improve SimPL results.

SimPL is highly competitive on ISPD 2005 benchmarks where it outperforms every placer available to us in binary both by solution quality and runtime. SimPL's advantage in runtime and solution quality over FastPlace3 and mPL6 grows on larger netlists. However, its most compelling advantages over prior state of the art deal with practical uses of placement in modern timing-closure design flows: (1) the reduced complexity of SimPL allows for fast implementation, parallel processing, and effective software maintenance; (2) the upper-bound placements facilitate tighter integration of timing and congestion optimizations into the global placement process, improving the speed and quality of physical synthesis.

The SimPL algorithm saw rapid adoption since its first publication at ICCAD 2010. At the ISPD 2011 placement contest, the winning team successfully implemented SimPL without having access to our source code. To our knowledge, at least two major EDA vendors are now using similar placement algorithms, and our own work with a state-of-

iterations, SimPL relies on anchors and pseudonets to ensure convergence.

[9]The concept of *area lookahead* was proposed in [43] for block-packing by nested bisection, where it checks if a given bisection admits a legal block packing in each partition. Area lookahead was not used in [43] to spread standard cells from dense regions.

the-art industry placer quickly produced significant improvements that will be discussed in later chapters.

The implementation of SimPL described in this chapter is designed for standard-cell layouts and does not yet handle movable macro-blocks. Our recent industry experience suggests that the majority of modern large-scale placement instances in practice do not require this feature, as their macro-blocks are fixed. However, mixed-size placement is useful for some mixed-signal SoCs, and we are addressing it in our ongoing work.[10]

Attempting to explain theoretically the strong performance of our placement algorithm, we have noticed similarities to *primal-dual* algorithms for convex [157] and combinatorial [21] optimization (see Chapter VI). Primal-dual methods maintain lower and upper bounds, expressed by primal and dual solutions that eventually converge to an optimal feasible solution. The interpretation of duality as swapping the problem's constraints for the objective function [157] is also consistent with our algorithm — *lookahead legalization corresponds to imposing a no-overlap constraint while relaxing the linear constraints that capture the global minimum of the quadratic wirelength objective*. The key to the success of primal-dual algorithms [21, 157] is the observation that alternating progress in *primal* and *dual* solutions, i.e., improving the cost of feasible solutions and tightening the constraints for low-cost solutions, typically leads to faster convergence compared to one-sided optimizations. This effect is empirically observed in Section 3.5 where SimPL is compared to pre-existing placement algorithms, all of which are one-sided.

[10] Academic placers typically introduce this feature in dedicated publications, rather than in the first publication describing the baseline algorithm.

CHAPTER IV

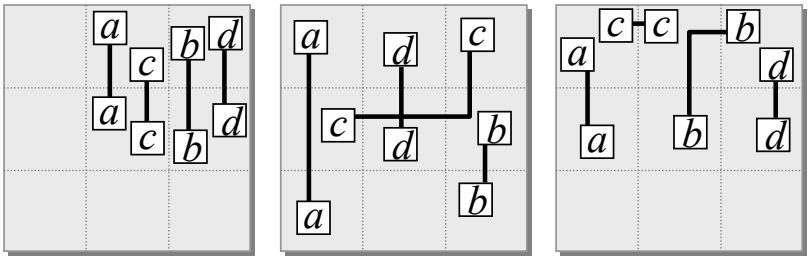
SimPLR: High-performance Routability-driven Placement

At advanced technology nodes, highly-optimized placements may lead to irreparable routing congestion due to inadequate models of modern interconnect stacks and the impact of partial routing obstacles. Additional challenges in routability-driven placement include scalability to large netlists and limiting the complexity of software integration. Addressing these challenges, we develop *lookahead routing* to give the placer advance, firsthand knowledge of trouble spots, not distorted by crude congestion models. We also extend global placement to (i) spread cells apart in congested areas, and (ii) move cells together in less-congested areas to ensure short, routable interconnects and moderate runtime. While previous work adds isolated steps to global placement, our SIMultaneous PLace-and-Route tool SimPLR integrates a layer-aware global router into a leading-edge, force-directed placer. The complexity of integration is mitigated by careful design of simple yet effective optimizations. On the ISPD 2011 Contest benchmarks, SimPLR outperforms every contestant on every benchmark by 2.03 times on average.

4.1 Introduction

Routability-driven placement can pursue several different optimization objectives, such as ensuring 100% routability, even at the cost of significant routing runtime. Alternatively, one can evaluate placements by a layer-aware global router with a short time-out, which nevertheless correlates with the final router (and is potentially based on the same software implementation). This intermediate objective is more amenable to optimizations in global placement because its quick evaluation facilitates a tight feedback loop. In other words, intermediate placements can be evaluated many times, allowing the global placer to make proper adjustments. As we show in Section 4.5, due to the correlation between the fast and the final router, resulting routability-driven placements may fare better even with respect to the former, more traditional objective. This approach offers additional benefits, such as early estimation of circuit delay and power in terms of specific route topologies. On the other hand, biasing the global placer away from HPWL to more sophisticated routability metrics may adversely affect the global placer's overall optimization capabilities. In other words, if HPWL increases too much, routability metrics will also increase.

In this chapter, we directly address the challenges of routability-driven placement based on our observations. **First**, we develop *lookahead routing*, which invokes a high-quality and fast 3-d global router, to quickly estimate routability. Since the produced routes can be used as a routing solution, our method can accurately and quickly report both congestion and routed wirelength. **Second**, to produce competitive placements in terms of both routed wirelength and HPWL, we integrate our lookahead routing into a flat, quadratic global placer, and enhance placement iterations by gently coercing cell locations and re-



TOF = 2
TWL = 4
D = 100%

TOF = 0
TWL = 7
D = 50%

TOF = 0
TWL = 5
D = 75%

Figure 4.1: The impact of placement density on routability, with bin capacity 2 and edge capacity 1. The dense, low-wirelength placement *on the left* is unroutable. The sparse, high-wirelength placement *in the center* is routable. The placement *on the right* is routable, with low wirelength and density.

lieve congestion while preserving interconnect length. We also account for routability in detailed placement, but rather than change the objective function [165], we *prohibit moves* that aggravate routability. In our global placement, cells in highly-congested areas are temporarily inflated to reserve whitespace during global placement. Traditionally, this has been accomplished by either cell bloating [20, 59, 120] during/after global placement, or by whitespace allocation [98,124,162] after placement. We observe that wirelength-driven global placers typically limit area utilization by a given amount through the entire layout based on target density. Therefore, in addition to *cell bloating*, we *dynamically adjust the target density* based on total routed wirelength.[1] This simple technique offers two benefits: it allows the placer to move cells in uncongested areas closer, and it preserves overall solution quality. **Third**, we develop a simultaneous place-and-route framework, where routability is addressed both at the global placement and detailed placement stages.

Our proposed methodology has several advantages with respect to routability and qual-

[1]Theoretically, target density can be changed on a per region basis, and this has been implemented in partitioning-based placers [20,124]. However, this feature is difficult to implement in force-directed placers, and, in our experience, unnecessary.

ity. First, since we use a global router to estimate congestion, the routes for all nets are known. Second, by enabling the global placer to directly redistribute whitespace in response to routing congestion, we establish a more precise feedback loop (compared to add-on techniques proposed previously). Third, since we use a variable target density, we are simultaneously optimizing for routability and solution quality by trading off routing demand for wirelength. As demonstrated in Figure 4.1, a placer using a lower target density typically produces a placement that is more likely to be routable, but has higher total wirelength. Conversely, a placer using a higher target density typically produces a placement that is less likely to be routable, but has lower total wirelength.

Our implementation uses the SimPL [84] global placer to quickly produce a tentative solution. Then, we apply lookahead routing by calling a modified version of the BFG-R [67] global router to estimate routing congestion and wirelength. We use this information in global placement by means of *cell bloating* and *dynamically adjust the target density*. After several iterations of global placement, performed using a modified SimPL placer, lookahead routing is invoked again. Such combined place-and-route iterations continue until convergence. Then, our modified version of the FastPlace-DP [112] detailed placer applies *congestion-aware detailed placement* to recover whitespace and improve routed wirelength while maintaining routability.

Our key contributions include:

- A method to control routability within the global placer while preserving solution quality by dynamically adjusting the global target density
- An effective cell-bloating technique by dynamically adjusting the width based on how 'hard' the design is

- A simultaneous place-and-route framework
- A congestion-aware detailed placement algorithm that moves cells only when this does *not* hurt routability
- Empirical results on the ISPD 2011 Contest benchmarks [143] that outperform every competitor on every benchmark

The remainder of this chapter is structured as follows. In Section 4.2, we classify the prior work on congestion-driven placers, and review the baseline software tools that we use. In Sections 4.3 and 4.4, we present the specific techniques that SimPLR uses to address routability within each design flow step. In Section 4.5, we present our empirical results, demonstrating the ideas presented in this chapter. Finally, in Section 4.6, we conclude this chapter.

4.2 Baseline Algorithms in Our Work

We briefly review the baseline placement and routing algorithms used in our work. We survey congestion maps and prior work on congestion-driven placement in Chapter 2.3.

SimPL (Chapter III) is a flat, force-directed global placer. It maintains both a lower-bound and an upper-bound placement solution; the final placement is generated when both solutions converge.[2] The upper-bound solution is generated by applying lookahead legalization, which is based on top-down geometric partitioning and nonlinear scaling. Using this information, the lower-bound solution is generated by minimizing the quadratic objective using the Conjugate Gradient method [127]. Here, $\vec{x}$ and $\vec{y}$ are coordinate vectors

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

of the cells' (x,y) locations, and $w_{i,j}$ represents the connectivity between cells i and j.

$$\Phi_q(\vec{x}, \vec{y}) = \sum_{i,j} w_{i,j} \left((x_i - x_j)^2 + (y_i - y_j)^2 \right) \tag{4.1}$$

FastPlace-DP [112] is a wirelength-driven detailed placer. It uses (i) cell clustering, (ii) global cell swapping, (iii) vertical cell swapping, and (iv) local reordering to improve wirelength. To determine which cells should be swapped, FastPlace-DP estimates the reduction in wirelength from swapping cells i and j by

$$gain(i, j) = \sum_{n \in N_i} (W_n - W'_n) - \sum_{n \in N_j} (W_n - W'_n) \tag{4.2}$$

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

BFG-R [67] is a global router based on Lagrangian relaxation. It decomposes multi-pin nets into two-pin subnets using MSTs and then iteratively routes all subnets until no violations are present. BFG-R prices each (sub)net by summing up the cost of used edges

$$cost(e) = base_e + \lambda(e) \times C(e) \times \rho(e) \tag{4.3}$$

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

4.3 Simultaneous Place-and-Route

In this section, we introduce a methodology for simultaneous place-and-route and discuss its components (see Figure 4.2). Given a placement instance, the baseline global placer quickly produces a tentative solution. Then, we apply *lookahead routing (LAR)* by

calling our global router to estimate routing congestion and wirelength. We use this information during global placement by means of *cell bloating* and *dynamically adjusting the target density*. After several iterations of global placement, where the placer "heals" the placement for wirelength, lookahead routing is invoked again, and such iterations continue until overflow stops improving. Then, the placer applies *congestion-aware detailed placement* to recover whitespace and improve routed wirelength while maintaining routability.

We achieve an initial placement solution once the wirelength gap between the upper-bound and lower-bound solutions is within 25% of the 10^{th} iteration's total wirelength (see Section 4.2). After cell bloating, we run four iterations of lookahead legalization. Our disjunctive convergence criterion checks for three conditions: (i) the overflow has improved less than 3% after two consecutive rounds of LAR, (ii) the total overflow is less than 1% of the total edge capacity, or (iii) the global placement timeout of 60 iterations.

4.3.1 Lookahead routing

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

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

$$Usage(net) = \sum_{e \in net} minSpacing(l_e) + minWidth(l_e) \tag{4.4}$$

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

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

$$nCap(e_l) = \frac{Cap(e_l)}{minSpacing(l_e) + minWidth(l_e)} \tag{4.5}$$

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

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

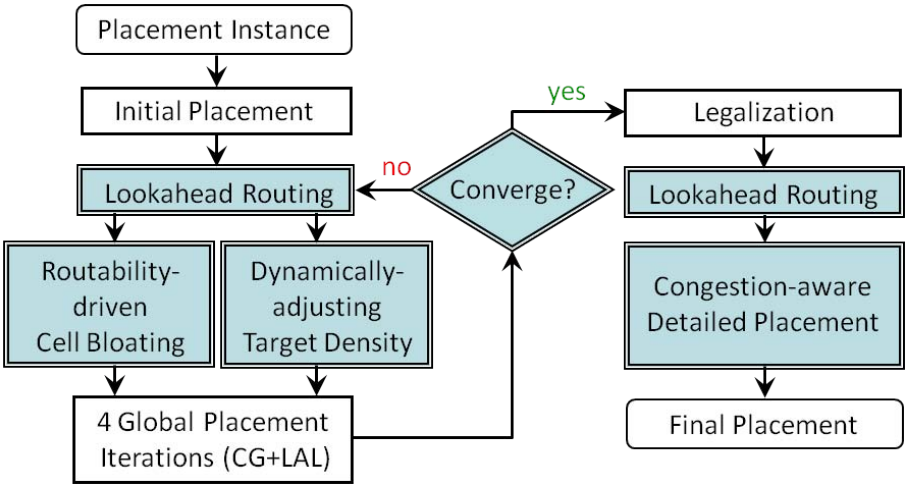


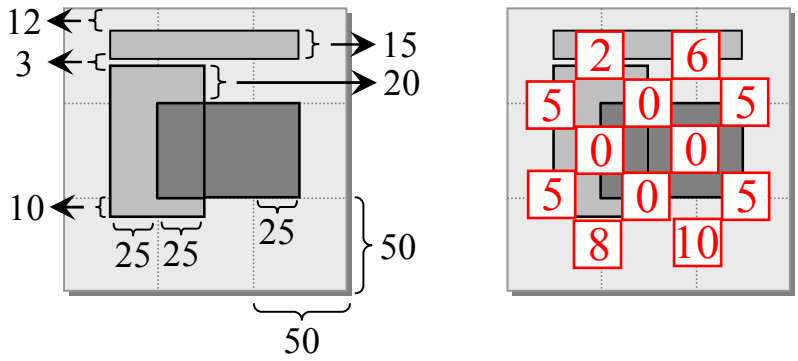
Figure 4.2: Our simultaneous place-and-route (SimPLR) flow. The baseline components are shown in transparent boxes, and the added routability-driven components are indicated by light-blue fill.

$$\upsilon(r_e) = \frac{dim(r_e)}{dim(e)} \tag{4.6}$$

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

$$nCap(e_l) = \sum_{r_e \in R_e} \frac{\upsilon(r_e) \times Cap(e_l)}{minSpacing(l_e) + minWidth(l_e)} \tag{4.7}$$

The calculation of normalized capacity in the presence of routing obstacles is demonstrated in Fig. 4.3. In the example, let the length of every edge be 50, let the lower left coordinate be (0,0). Let the original edge capacity be 40, and let the minimum spacing plus the minimum width of this layer be 4. Since the vertical edge (50,0)-(50,50) has coordinates (50,40)-(50,50) blocked off, it only has $(50-10)/50 = 80\%$ usable capacity. Since there is only one non-blocked region, the normalized capacity would be $\frac{(80\% \times 40)}{4} = 8$. Similarly, the horizontal edge (50,50)-(100,50) has no usable capacity, as it is entirely



$dim(e_v) = dim(e_h) = 50$ $\quad origCap(e_v) = origCap(e_h) = 40$, $minWidth(e) + minSpacing(e) = 4$

Figure 4.3: Accounting for routing blockages, where $dim(e) = 50$ for each edge. Two of three routing blockages overlap. On the left, the lengths of each routing blockage and non-blocked region are shown. On the right, the normalized capacities are calculated for each edge. Here, the original capacity of each edge is 40, and each net on this layer uses 4 tracks. With no blockages, an edge has a normalized capacity of 10.

blocked off, so its normalized capacity is 0.

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

4.3.2 Congestion-based cell bloating

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

$$C(g(x,y)) = \frac{nUsage(g(x,y))}{nCap(g(x,y))} \tag{4.8}$$

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

The usage at each gcell is defined as

$$Usage(g(x,y)) = \max(nUsage(e(x \pm 1, y), nCap(x \pm 1, y))) + \max(nUsage(e(x, y \pm 1), nCap(x, y \pm 1))) \tag{4.9}$$

and the capacity at each gcell is defined as

$$Cap(g(x,y)) = nCap\left(e(x \pm 1, y)\right) + nCap\left(e(x, y \pm 1)\right) \tag{4.10}$$

where $nUsage$ is the normalized usage for edge e, and $nCap$ is the normalized capacity for e. Therefore, if $C(g(x,y)) > 1$, then $g(x,y)$ is considered congested. If at least one of the neighboring edges is congested, then the gcell is considered congested.

Then, for every cell in each congested gcell, we apply cell bloating by setting the cell's new width to

$$\max(width(cell) + 1, 1 + \theta \cdot \Lambda(cell) \cdot C(g(x,y)) \cdot deg(cell)) \tag{4.11}$$

where $width$ is the current width of $cell$, θ is an adaptive function (described below), Λ is the number of times $cell$ has been in a congested gcell, and deg is the cell's degree.

Our cell bloating approach is inspired by CRISP [120], but differs in three major ways. First, we apply cell bloating *during global placement*, while CRISP bloats cells *after placement*. We can therefore perform large-scale changes and, in our experience, our placer better adjusts to bloated cells, resulting in a smaller wirelength penalty. Second, we use *gcell-centric congestion* estimation, while CRISP uses *edge-centric congestion* estimation with a pin-density map. Our style of congestion estimation improves integration with a global router. Pin density has been a popular estimation technique for designs

with relatively few metal layers. However, with modern 9+ layer interconnect stacks, it primarily affects the success of *detailed routing*, which is orthogonal to our work.[3]

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

$$\theta(G) = \max\left(0, \alpha \cdot \eta(G) \cdot \xi(G) + \beta\right) \tag{4.12}$$

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

$$\eta(G) = \sum_{g \in G} \frac{Usage(g)}{Cap(g)} \tag{4.13}$$

and $\xi(G)$ as

$$\xi(G) = \frac{TOF(G)}{TCap(G)} \tag{4.14}$$

where $OF(G)$ and $Cap(G)$ are the respective total overflow and total capacity of all gcells in G. In our implementation, we empirically determined the values $\alpha = 0.017$ and $\beta = -0.01$ based on numerical regressions (no benchmark-specific tuning was performed).

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

4.3.3 Dynamic adjustment of target density

Target density (utilization) is one of the most critical parameters to trade off routability for wirelength in the final placement. However, finding the best target density for routability-driven placements remains an open problem. Unnecessarily high target density leads to better HPWL, but may also cause routing failures [4]. Lower target density, on the other hand, may increase the overall routed wirelength, which would lead to longer detours and consume more routing resources. Initially, we set the target density as

$$\gamma = D_{ut} + \min(\max(\gamma_0 - D_{ut}, 0\%), \omega_D) \tag{4.15}$$

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

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

$$\gamma = \frac{area(C_m)}{area(D) - area(C_f)} + \phi \tag{4.16}$$

where C_m is the set of movable cells, C_f is the set of fixed cells, D is the design, $area$ returns the total area of input (bloated cells included), and ϕ is a constant that increases every time LAR reports an increase in routed wirelength. In our implementation, ϕ is initially $\gamma_0 - D_{ut}$, and increases by 1% when wirelength increases.

4.4 Congestion-aware Detailed Placement

Traditional wirelength-driven detailed placement may pack cells in areas that are difficult to route. In the context of the FastPlace-DP algorithm, we modify both global cell swapping and vertical cell swapping to be *congestion-aware* (see Algorithm 2) in two ways. First, we only allow cells to move when such moves do not harm routability. Second, we encourage cells to move out of congested areas.

Algorithm 2 Congestion-aware Detailed Placement

1. C_m = Set of all movable cells
2. G_c = Set of all congested gcells
3. C(i) = Congestion in the position cell i
4. **foreach** $c_i \in C_m$
5. find the optimal region R_i of c_i
6. find b_{swap}, the *benefit of swap* with a cell $c_j \in R_i$
7. find b_{move}, the *benefit of move* to a space $s \in R_i$
8. **if** ($c_i \notin G_c$ && $c_j \notin G_c$)
9. **if** ($b_{swap} \geq b_{move}$)
10. perform_swap(c_i, c_j, ($b_{swap} > 0$))
11. **else**
12. perform_swap(c_i, s, ($b_{move} > 0$))
13. **else if** ($c_i \in G_c$ && $c_j \notin G_c$)
14. perform_swap(c_i, s, *true*)
15. **else if** ($c_i \notin G_c$ && $c_j \in G_c$)
16. perform_swap(c_i, c_j, ($deg(c_i) < deg(c_j)$))
17. **else**
18. if (C(c_i) > C(c_j))
19. perform_swap(c_i, c_j, ($deg(c_i) > deg(c_j)$))
20. else
21. perform_swap(c_i, c_j, ($deg(c_i) < deg(c_j)$))
22. **end foreach**

The subroutine perform_swap(c_i, c_j, $pred$) swaps two cells c_i and c_j if $pred$ is true.[4] For each movable cell c_i, we consider its best swap (with c_j) or move (with empty space s). If both actions result in positive gain, and both are in non-congested areas, then we

[4]To move a single cell c, a cell can swap with an empty location s.

revert to wirelength-driven decisions. If c_i is in a congested region and c_j is not, then we can improve routability by moving it to s. If c_i is not in a congested region, but c_i is and has fewer pins than c_j, we can potentially improve routability in subsequent moves if we decrease the number of routes that go through the congested region. Similarly, if both cells are in congested regions, then we only swap them if $deg(c_j) < deg(c_i)$. This ensures that the detailed placer does not harm routability.

4.5 Empirical Validation

Our implementation was written in C++, compiled with g++ 4.4.3, and validated on a 3.00 GHz Intel Core 2 CPU X9650 Linux workstation. We modified and integrated the (i) SimPL global placer [84], (ii) BFG-R global router [67], and (iii) FastPlace-DP detailed placer [112]. Significant changes were made to all three tools, and new algorithms were added, as described in Sections 4.3 and 4.4.

The evaluation of placement solutions was performed by coalesCgrip [41], which was mandated by the ISPD 2011 Routability-driven Contest [143]. coalesCgrip was compiled with gcc 4.4.1, as specified by the contest organizers. Its runtime limit was set to 300 seconds for initial routing and 900 seconds for rip-up and reroute (RRR), which makes results machine-dependent. Therefore, we downloaded all placements produced by the top contestants, and reevaluated them on our servers.

Routability. As shown in Table 4.5, our implementation consistently reduces the total overflow across all benchmarks. Compared to baseline wirelength-driven placer SimPL, we improve total overflow by 4.52 times on average. Compared to the best-reported results from the ISPD 2011 Routability-driven Contest, we improve total overflow by 2.03

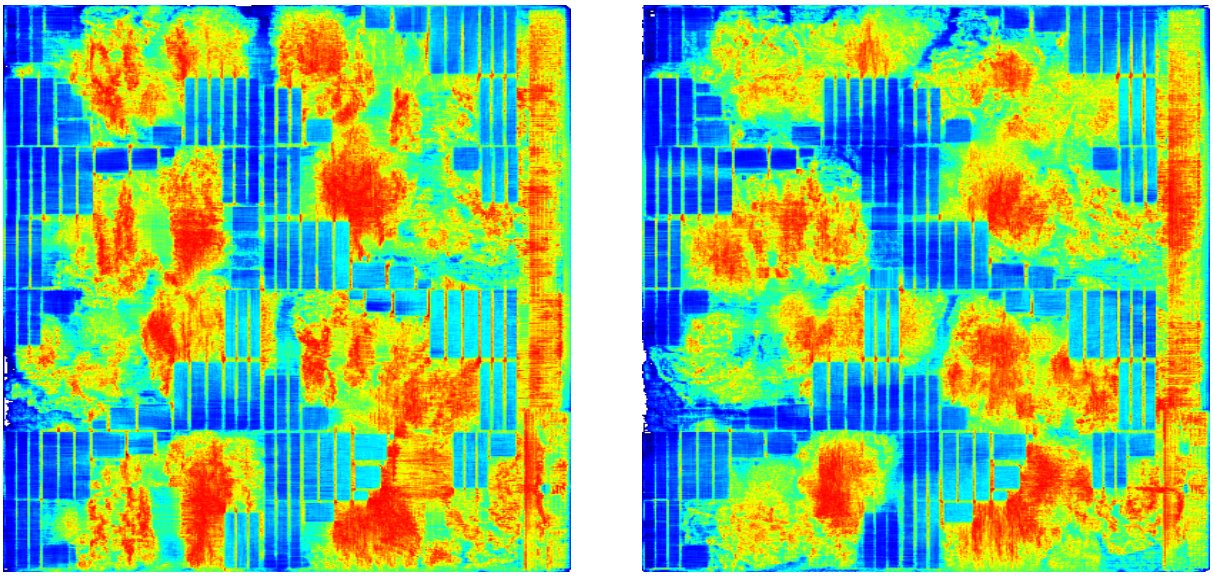


Figure 4.4: Congestion maps for the benchmark SUPERBLUE15 for the best-reported placement at the ISPD 2011 Contest (left) and SimPLR (right).

times on average. We were also able to produce placements with the smallest overflow for all eight circuits in the ISPD 2011 benchmark suite (no parameter tuning to specific benchmarks was employed). As shown in Figure 4.5, our congestion map on *superblue15* is competitive with that produced by the top contestants of the contest.

Runtime. We compared SimPLR to SimPL runs with the same initial target density (50%). In our single-threaded implementation, SimPLR takes about twice as long as SimPL. This is due to invoking lookahead routing 3-5 times per benchmark, and consuming 27-58% of the total runtime, averaging at 47.88%.

Congestion-aware Detailed Placement. We measure the (i) recovered HPWL, (ii) recovered routed length, and (iii) total overflow improvement using our congestion-aware detailed placement (Ca-DP), as compared to FastPlace-DP [112]. On average, Ca-DP did not change the HPWL or the routed wirelength on the design, and improved overflow by 4%. Our approach consistently reduced the overflow with the exception of *superblue10*.

BENCHMARK		SimPL			Best in Contest		SimPLR		
		RtWL	OF	Runtime	RtWL	OF	RtWL	OF	Runtime
SUPERBLUE1	847K	14.33	1468	23.89	14.70	108	14.48	0	51.69
SUPERBLUE2	1.01M	27.22	1203412	37.87	30.77	797898	29.20	740050	108.30
SUPERBLUE4	500K	10.55	12744	7.54	10.86	85538	10.68	18444	24.79
SUPERBLUE5	772K	16.99	262358	23.53	17.29	126186	16.98	121894	51.84
SUPERBLUE10	1.13M	26.19	479352	33.68	25.16	616742	26.69	567780	73.34
SUPERBLUE12	1.29M	19.44	1998998	35.18	22.89	415428	22.58	181350	43.32
SUPERBLUE15	1.12M	16.24	335756	24.21	17.91	125936	17.07	49286	43.33
SUPERBLUE18	483K	10.62	178398	14.36	9.84	31440	10.63	21020	21.38
Average		0.96×	4.52×	0.51×	1.00×	2.03×	1.0×	1.0×	1.0×
Geometric mean		0.96×	2.79×	0.49×	1.00×	1.78×	1.0×	1.0×	1.0×

Table 4.1: Routed wirelength (RtWL, ×10e6), routing overflow (OF), and runtime (in minutes) on the ISPD 2011 routability-driven contest benchmark suite. The placements were evaluated by *coalesCgrip* [41].

This confirms that our approach not only maintains the original quality of the wirelength-driven detailed placer, but also improve routability.

Implementation. In SimPLR, we use BFG-R for lookahead routing instead of the evaluation router for two reasons. Empirically, our router is able accurately predict the areas of congestion reported by coalesCgrip. Since our strong results are achieved *without* running coalesCgrip, it is likely that SimPLR does not require the knowledge of a specific downstream router. Though we do not advocate using a different router than the evaluation router, we show that this is possible.

Since lookahead routing is used for estimation, it produces a *valid* routing solution, where all nets are routed, but not necessarily legal or well-optimized for wirelength. In our implementation, we simply limited the number of iterations that BFG-R uses. However, it is possible to run a full router during this process. In particular, the produced routes can be used for other applications, such as targeting specific nets for timing optimization.

BENCHMARK		FastPlace-DP				Ca-DP			
		HPWL	RtWL	OF	Runtime	HPWL	ΔRtWL	ΔOF	Runtime
SUPERBLUE1	847K	277.03	14.45	0	5.37	279.01	0.376	0	9.83
SUPERBLUE2	1.01M	657.03	29.09	782348	19.22	660.09	-0.195	-42298	32.06
SUPERBLUE4	600K	231.78	10.71	22192	2.96	231.44	-0.336	-3748	4.62
SUPERBLUE5	772K	354.23	17.02	139012	5.58	355.05	-0.386	-17118	9.68
SUPERBLUE10	1.13M	586.62	26.48	556678	7.99	592.18	0.113	11102	18.26
SUPERBLUE12	1.29M	376.59	22.7	293516	7.71	377.27	-0.119	-112166	13.33
SUPERBLUE15	1.12M	337.04	17.04	56866	6.60	337.96	0.128	-7580	8.43
SUPERBLUE18	483K	165.09	10.64	23708	2.92	165.75	-0.125	-2688	4.44
Average		1.00×	1.00×	1.04×	0.60×	1.00×	1.00×	1.00×	1.00×

Table 4.2: The impact of our congestion-aware detailed placement on HPWL(×10e6), routed wirelength (×10e6), and overflow (OF) on the ISPD 2011 Routability-driven Contest Benchmark Suite. The runtime is reported in minutes. The placement solutions are evaluated by *coalesCgrip* [41].

4.6 Summary and Conclusions

Tight integration of major CAD tools is sometimes frowned upon in the industry because it may sharply increase software complexity, introduce subtle discrepancies and complicate software maintenance. However, such integration is highly sought in place-and-route, where high-performance global placers often generate hard-to-route solutions, creating unnecessary complications for downstream tools. The strategy pursued in our work is to give the placer advance, firsthand access to tentative net routes and resulting actual congestion maps (rather than crude estimates), as well as the ability to respond early and often. We believe that our proposed integration of global routing into global placement, *based on lookahead routing of upper-bound placements in the SimPL* algorithm, offers a particularly promising and "clean" path to effective simultaneous place-and-route. By communicating through a lightweight interface, the placer and the router quickly exchange multiple updates to cell locations and net routes, while maintaining the software infrastructure separated. This framework facilitates several further extensions, especially

in timing optimization, where the placer has early and direct access to actual routes to improve the accuracy of delay estimation. Empirical data show that our approach is able to effectively improve routability, reducing total overflow by 2.03 times on average compared to the best-reported results at the ISPD 2011 contest.

CHAPTER V

MAPLE: Multilevel Adaptive PLacEment for Mixed-size Designs

Extending the SimPL placement algorithm introduced in Chapter III, we propose a new multilevel framework for large-scale placement called MAPLE that respects utilization constraints, handles movable macros and guides the transition between global and detailed placement. In this framework, optimization is adaptive to current placement conditions through a new density metric. A novel component called Progressive Local Refinement (ProLR) helps mitigate disruptions in wirelength that we observed in leading placers. Our placer MAPLE is implemented using EDA infrastructure at IBM and outperforms all previously published empirical results — RQL, SimPL, mPL6, NTUPlace3, FastPlace3, Kraftwerk and APlace3 — across the ISPD 2005 and ISPD 2006 benchmarks, in terms of official metrics of the respective contests.

5.1 Introduction

Large-scale placement remains one of the most influential optimizations in interconnect-driven physical design and physical synthesis [10]. Despite the long history of research, three ISPD contests on placement have shown that recent algorithms achieve sizable gains

over prior state of the art [107]. The ISPD 2011 routability-driven placement contest [143] has demonstrated that the choice of the wirelength-driven global placement engine is paramount even in multiobjective placement — two of the top three teams relied on the high-quality SimPL framework [86], including the contest winners, who reimplemented SimPL without having access to the original source code [57]. Yet, no placer dominated across the entire benchmark set, indicating possible improvements. Such improvements are described in this chapter, although our work is orthogonal to and compatible with the innovations developed for the ISPD 2011 contest [57, 64, 85].

In this chapter, we develop MAPLE — a multilevel force-directed placement algorithm that pioneers key algorithmic components and a more effective way of combining individual components into a reliable multiobjective optimization. MAPLE generates a coarsest-level placement by a variant of the SimPL algorithm (Chapter III) but also employs multilevel extensions reinforced by our new Progressive Local Refinement (ProLR).[1] This combination enhances tradeoffs between wirelength and module density. Compared to recent literature, our implementation improves solution quality with reasonable runtimes. The improvement on ISPD 2006 benchmarks is particularly encouraging because it demonstrates that MAPLE not only reduces the wirelength but also avoids highly concentrated placements, thus promoting routability and providing greater flexibility for timing optimization transforms. Note that the original SimPL algorithm was not evaluated with utilization constraints of the ISPD 2006 benchmark suite and could not handle movable macros present in those benchmarks. At a more conceptual level, our work explores limits to optimization imposed by noise inherent in analytic placement algorithms. After

[1] The implementation used in this chapter was written from scratch.

studying sources of this noise, we develop techniques to avoid noise or suppress it, which consistently improve end results beyond the best reported in the literature.

Our key contributions include:

- A study of obstacles to extending analytic placement with multilevel techniques. We observe that straightforward extensions cause disruptions between successive optimizations during global placement.
- A key insight to combine unclustering with two-tier Progressive Local Refinement (ProLR) so as to ensure graceful transitions between optimizations at different cluster levels. Optimization adapts to current wirelength/density tradeoffs, which we track by a newly developed metric — ABU_γ.
- A placement algorithm (MAPLE) that relies on SimPL iterations, but augments them with two-level clustering and ProLR. MAPLE guides the transition from global to detailed placement to avoid unnecessary disruptions. This guidance allows MAPLE to derive the final placement from the lower- rather than the upper-bound placement as in the original SimPL, enhancing solution quality.
- Extensions of the MAPLE algorithm to handle movable macros. This includes extending the SimPL algorithm and dealing with macros during refinement.
- Empirical evaluation against best published results on ISPD 2005 and ISPD 2006 benchmarks using official metrics. MAPLE consistently outperforms all leading-edge placers described in the literature.

The remainder of this chapter is structured as follows. Section 5.2 presents background and prior art. Section 5.3 analyzes disruptions during multilevel placement optimization

that undermine solution quality. In Sections 5.4 and 5.5, we present the MAPLE algorithm and specific techniques to ensure graceful transitions between successive optimizations. Section 5.6 describes extensions of the MAPLE algorithm to handle movable macros. Section 5.7 empirically validates our ideas and algorithms. Section 5.8 summarizes the results of this chapter.

5.2 Foundational Algorithms

FastPlace-Global [140] is a force-directed quadratic placer with two-level Best-choice clustering [9]. It relies on a hybrid (star-clique) net model[2] and employs *cell shifting* to spread the modules during the early stages of placement flow. The *Iterative Local Refinement (ILR)* technique is applied after quadratic optimization to reduce HPWL and spread the modules (see Section 5.5). **RQL [141]** extends FastPlace-Global by limiting spreading forces (*force-vector modulation*). **FastPlace-DP [112]** is a wirelength-driven detailed placer based on (i) single segment cell clustering, (ii) global cell swapping, (iii) vertical cell swapping, and (iv) local reordering.

SimPL (Chapter III) is a flat, force-directed global placer. It maintains a lower-bound and an upper-bound placement and progressively narrows the displacement between the two. The final solution is derived from the upper-bound placement when the two bounds converge. The upper-bound placement is generated by *lookahead legalization (LAL)*. Applying the upper-bound placement as fixed-points, the lower-bound placement is generated by minimizing the quadratic objective using the CG method. Unlike FastPlace-Global and RQL, the SimPL algorithm relies on the Bound2Bound net model [134].

[2]The numerical equivalence of the clique model and the star model with a star node was pointed out in [92] and proven in [81].

5.3 Analysis of disruptions during analytic optimization

State-of-the-art algorithms for placement integrate multiple optimization steps, which sometimes target different objectives. Poor coordination between successive steps may cause radical changes in intermediate placements. These changes become disruptive when they reverse improvement obtained by previous steps, increasing overall runtime and undermining final solution quality. We now investigate the sources of disruptive changes between successive stages of analytic placement.

Unclustering. In multilevel global placement algorithms, placement iterations after unclustering often include changes to the optimization objective as well as the netlist. This may abruptly increase wirelength as illustrated in [78, Figure 4] for APlace. The authors state that "*Clustering helps to spread cells more quickly, but wirelength is impaired during cell expansion. It is clearly seen from the figures that when wirelength weight is decreased and the conjugate gradient optimizer restarts, discrepancy drops sharply and wirelength is often increased at first and then refined during the optimization*". However, in contrast to our observation in Section 5.5, the authors claim that when both discrepancy (overflow) and wirelength change slowly, they obtained a near stable suboptimal solution, in which additional iterations did not further reduce discrepancy and wirelength without a major change to the parameters.

Transition to the HPWL objective. FastPlace [140] and RQL [141] use the ILR iterations to recover HPWL after quadratic optimization and before detailed placement. The ILR iterations include bin resizing over wide ranges to allow large moves across the placement region [107, Chapter 8]. Moreover, each bin maintains a bin-specific utilization weight

$0 \leq \theta \leq 1$, which changes depending upon the current bin's utilization. As history accumulates on dense bins over iterations, ILR increasingly penalizes such bins and allows abrupt moves to decrease local density (Figure 5.1). The density metric ABU_{10} is defined in Section 5.4.2.

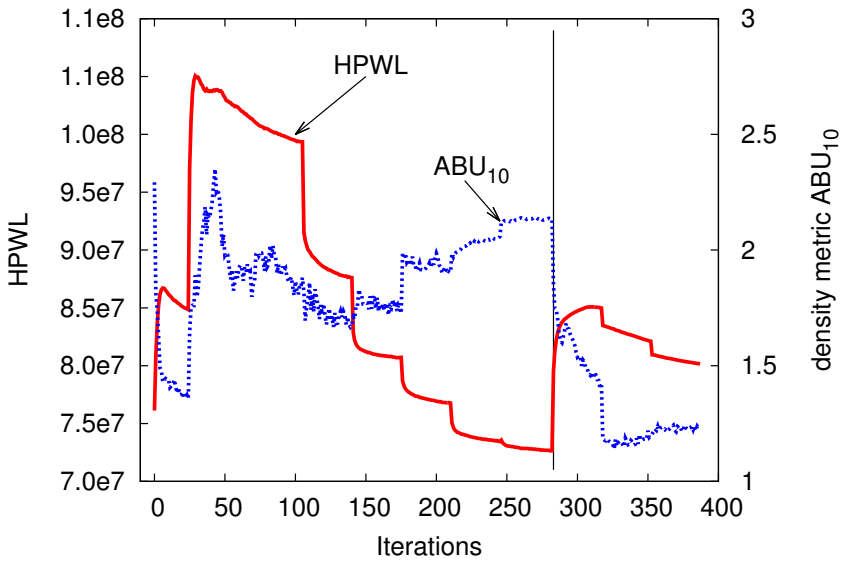


Figure 5.1: Progressions of wirelength and the density metric ABU_{10} over the ILR iterations on ADAPTEC1. Unclustering is marked with a vertical line. ILR disruptively improves ABU_{10} and increases the wirelength. Each ILR iteration traverses all movable modules once.

Hand-off to detailed placement. Recall that the SimPL algorithm maintains two placements throughout its iterations, and legalization is invoked on the *upper-bound* placement, when the lower- and upper-bound placements are reasonably close. The lower-bound placement within SimPL is analogous to module locations maintained by other algorithms. Instead of using the upper-bound, invoking (full) legalization on the lower-bound placement should be potentially better in preserving wirelength optimized by the linear system solver. However, these placements typically exceed target utilization and undergo significant changes during full legalization (Figure 5.2). Despite local improvement in wire-

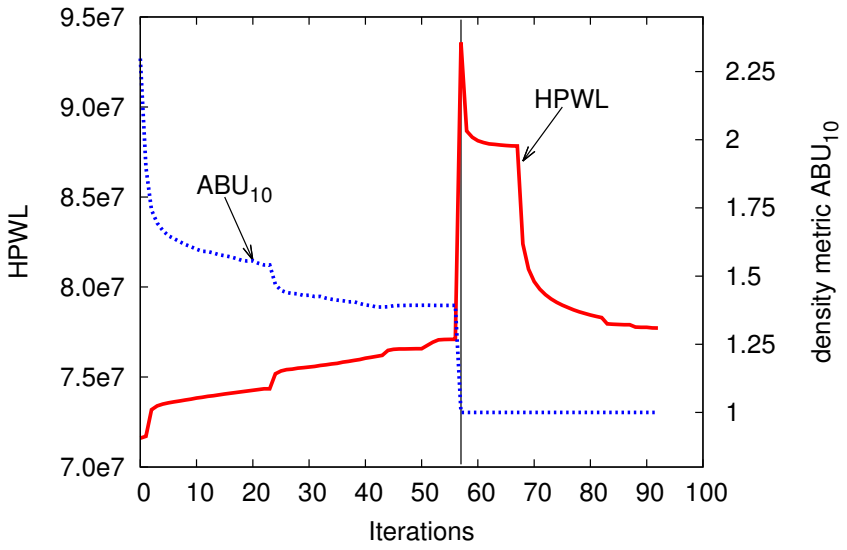


Figure 5.2: Progressions of wirelength and the density metric ABU_{10} over FastPlace-DP iterations on ADAPTEC1. The start of detailed placement is marked with a vertical line. Placements with high utilization undergo significant changes as full legalization completes.

length during detailed placement, such abrupt changes are detrimental to solution quality in terms of wirelength, routing congestion and timing.

Strategies for mitigating disruptions. Disruptions during analytic optimization can be mitigated by ensuring gradual transitions between successive optimizations. With this in mind, we develop a new use of placement metrics to make these transitions more adaptive to the actual module distribution and interconnect characteristics. **(1)** the overall placement flow is modified at the points where the objective function abruptly changes, as identified in the above analysis — before/after unclustering, and before detailed placement. We introduce a new intermediate stage that optimizes a linear combination of the preceding and succeeding objective functions, while gradually modifying parameters to ensure smooth transition between the objectives. **(2)** At each substage, we seek near-monotone improvement of either wirelength or module density in a predictable manner without disrupting the

other objective. **(3)** Specifically, each intermediate stage prohibits abrupt cell movement and significant changes in key objective functions. Small moves are encouraged instead, as this smoothens changes in wirelength and module density. **(4)** Weighting is adaptively updated according to a new placement metric.

5.4 Multilevel Adaptive Placement

We developed our global placement algorithm to address or circumvent the pitfalls in prior art discussed above. This technique consists of three phases: clustering, top-level (coarsest-level) placement iterations, and Progressive Local Refinement (ProLR) used in conjunction with unclustering (Algorithm 3). We apply Best-choice clustering [9] until the number of clusters is reduced to half the size of the flat netlist. Top-level placement iterations perform quadratic optimization on a coarsened netlist and globally regulate module densities over the placement region while moderating wirelengh increase. We adopt a variant of the SimPL algorithm [86] for this phase. The ProLR technique discussed in Section 5.5 improves both wirelength and module density before and after unclustering. Section 5.7.3 gives an outlook for using more than two levels of clustering.

5.4.1 Top-level placement iterations

Top-level placement for the coarsest netlist is performed by the SimPL force-directed placement. It generates lower- and upper-bound placements at each iteration and reduces the displacement gap between the two upon convergence. In contrast to the original SimPL algorithm, MAPLE chooses the last lower-bound placement as a final solution of quadratic placement iterations. This choice is based on our observation that our implementation of

Algorithm 3 Multilevel Adaptive PlacEment (MAPLE)

Phase 0: Clustering of Standard Cells
  N_0 = number_of_modules in flat netlist
  while number_of_clusters $> N_0$ / 2.0 **do**
    cluster netlist using the Best-choice clustering algorithm
  end while

Phase 1: Top-level Placement Iterations (SimPL extended)
  initial HPWL optimization
  while ABU_{10} of lower-bound placement $>$ threshold **do**
    transform the lower-bound placement into an upper-bound
    placement by Extended Lookahead Legalization (E-LAL)
    fix movable macros upon stabilization (Section 5.6)
    update *pseudopin* locations and *pseudonet* weights
    in the linear system [86]
    solve the updated linear system using
    the preconditioned CG method
  end while

Phase 2: Refinement for Mixed-size Netlists
  determine parameters for ProLR
  perform ProLR-w and ProLR-d optimizations
  legalize and fix all movable macros **// the end of Phase2a**
  while number_of_modules $< N_0$ **do**
    uncluster the netlist
    place unclustered cells side by side
  end while
  recalculate parameters for ProLR
  perform ProLR-w and ProLR-d **// the end of Phase2b**

SimPL in MAPLE does not completely close the gap between lower and upper bounds. Also, given that *lookahead legalization* [86] is unaware of wirelength objectives, the upper-bound placements are likely to suffer suboptimality. On ISPD 2005 benchmarks, MAPLE typically exhibits a gap of 5.63% to 13.89% between lower and upper bounds at its final iterations. However, lower-bound placements typically exhibit higher module density. To address this challenge, we improve lower-bound placements using local-search techniques, as described in Section 5.5.

5.4.2 A placement density metric - ABU_γ

We now explore density metrics during global placement, which provide insights into the quality of module spreading in intermediate placements and estimate wirelength impact of legality enforcement. Based on such a metric, the global placer can adaptively adjust its parameters depending on how concentrated the placement is, as described in Section 5.5.[3] To this end, we propose a new density metric, ABU_γ — average bin utilization of the top $\gamma\%$ densest bins excluding bins fully occupied by fixed macros. Given that the top $\gamma\%$ densest bin are averaged,[4] this metric reflects the non-uniformity of module distribution (Figures 5.1 and 5.2). Compared to overflow-based metrics, ABU_γ provides a more intuitive, cross-design perspective into the quality of module spreading. Monitoring density along with wirelength during placement enables comparisons of different parameter settings and even different placers (Figure 5.3). Such comparisons accelerate algorithm development and evaluation.

[3]Little is published on density metrics for global placement. Metrics based on *averaged overflow* (including *scaled-overflow per bin* in the ISPD 2006 contest) often fail to capture uneven module distribution. The *maximum utilization* metric leads to pessimistic estimation in the presence of many fixed modules.

[4]In our experiments $\gamma = 10\%$ and the square grid bins have 6 standard-cell heights on the side.

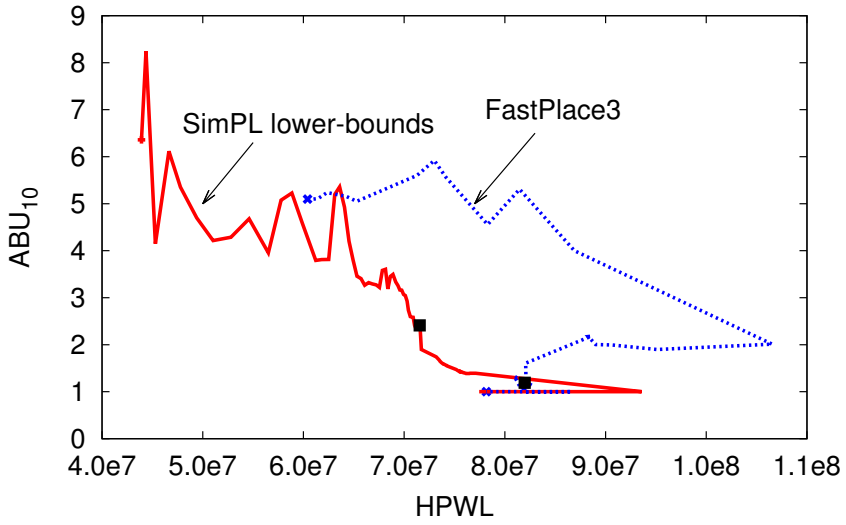


Figure 5.3: Progression of the density metric ABU_{10} versus wirelength, comparing SimPL lower-bounds (w/ FastPlace-DP) and FastPlace3 on ADAPTEC1. Steeper slope and datapoints closer to the origin indicate better tradeoffs. Each square box indicates the beginning of detailed placement.

5.5 A Methodology for Graceful Optimization in Placement

After quadratic optimization, placements typically exceed the target utilization in many regions, and their HPWL can be improved without increasing max module density. Furthermore, unclustering traditionally counts on subsequent quadratic placement and can be simple-minded in placing modules within clusters. MAPLE improves this situation by using ProLR — a two-tier technique to reduce wirelength and max module density. ProLR adopts single iterations of ILR [140, 141] — **L***ocal* **R***efinement (LR)* — as a baseline and a vehicle for placement modification. While ILR tends to be disruptive, ProLR promotes gradual transitions via (1) limited bin resizing, (2) *Explicit Bin-Blocking (EBB)*, (3) careful scheduling of utilization weights (θ) between wirelength and module density, and (4) optimizing one objective at a time, while limiting changes to other objectives; such optimizations are alternated.

Bin sizing. ILR and ProLR use regular bin structures and greedily move modules between adjacent bins based on Formula 5.1. Unlike in ILR, the bins in ProLR are small and remain unchanged during each invocation of LR. Each bin is 5 times the average movable-module area (bins shrink after unclustering). This restricts moves in ProLR.

Explicit Bin-Blocking (EBB) makes local-refinement moves less disruptive. The technique consists of two components: EBB^+ and EBB^-. EBB^+ stops the inflow of modules to some bins (when such moves are expected to be harmful), while EBB^- stops the outflow of modules from some bins and encourages the inflow of modules into these bins. Therefore, EBB^+ is applied to a handful of bins to limit density, while EBB^- is applied to a larger set of bins to attract modules from remaining bins (the density of these bins may decrease).

Joint optimization of density and wirelength. Local refinement moves individual modules based on the linear combination of improvements in HPWL and density.

$$\text{Score}(m) = \alpha \cdot \Delta_{HPWL} + \beta \cdot \theta \cdot \Delta_{density} \tag{5.1}$$

where θ is the *utilization weight*, and α and β are normalizing coefficients [107, Chapter 8]. In FastPlace and RQL, bin-specific θ_b values are managed after they are reset to values $0.4 \leq \theta \leq 0.6$ when ILR iterations start at each level.

Existing move-based algorithms for optimizing (i) max density and (ii) HPWL use effective techniques for finding highest-gain moves. Yet, no known algorithms are currently known for directly finding the best moves with respect to Formula 5.1. ProLR inspects best moves for each objective and select those that do not harm the other objective. ProLR performs two simpler optimizations ProLR-w and ProLR-d, which optimize wirelength

and module density, respectively. To smoothen placement changes, utilization weight (θ) starts from a small value $\theta_w^0 = 0.1$ for ProLR-w with a coarsened netlist, and θ_{step}^0 is found via a monotonic function

$$\theta_{step}^0 = f(\Upsilon_{target} - \Upsilon_{design}) \tag{5.2}$$

When the difference between *design utilization* (Υ_{design}) and *target utilization* (Υ_{target}) is small, placement iterations should aggressively reduce density, which is achieved by using a large θ_{step}^0 (greater emphasis on spreading in LR). On the other hand, a wider gap between the two justifies a greater weight for wirelength, and the best wirelength is often achieved by using a small θ_{step}^0 (greater emphasis on wirelength in LR). To implement Formula 5.2, MAPLE uses a step function that distinguishes three different cases: (i) emphasis on wirelength optimization, (ii) no bias, and (iii) emphasis on spreading. Given that Υ_{design} is fixed, the step function only depends on Υ_{target}, which is typically chosen by the designer. Assuming fixed-outline placement ($\Upsilon_{target} \geq \Upsilon_{design}$),

$$\theta_{step}^0 = \begin{cases} 0.0250, & \text{if } \Upsilon_{target} - \Upsilon_{design} \geq 0.5 \\ 0.0275, & \text{if } \Upsilon_{target} - \Upsilon_{design} \geq 0.05 \\ 0.0375, & \text{if } \Upsilon_{target} - \Upsilon_{design} < 0.05 \end{cases} \tag{5.3}$$

The utilization weight for ProLR-w with a flat netlist, θ_w^1 is determined as $\theta_w^1 = \theta_d^{M-1}$ where M is the number of ProLR-d invocations performed for the coarsened netlist. The θ_d^k values in the k-th invocation of ProLR-d are determined by

$$\theta_{step}^k = \theta_{step}^{k-1} \cdot (1 + \frac{ABU_{10}}{100\Upsilon_{target}}) \tag{5.4}$$

$$\theta_d^k = \theta_w^{k/M} + \theta_{step}^k \quad \forall k \in \{0, M\} \tag{5.5}$$

$$\theta_d^k = \theta_d^{k-1} + \theta_{step}^k \quad \forall k \notin \{0, M\} \tag{5.6}$$

ProLR-w improves placement wirelength while maintaining the initial module density

distribution. As ProLR-w begins, bin-specific θ_b are reset to θ_w^0 for the clustered netlist and to θ_w^1 for the flat netlist. These values are updated throughout the LR iterations of ProLR-w. Given that ProLR-w maintains θ_b over the entire 300 LR iterations, it closely resembles the use of ILR in FastPlace [140]. However, ProLR-w prohibits abrupt cell movement and significant changes in placement by (1) EBB^+ for bins whose utilization exceeds ABU_{10} and (2) keeping small bin sizes. ProLR-w terminates when ABU_{10} of the current placement exceeds the initial ABU_{10}. Otherwise, ProLR-w continues until there is no improvement in wirelength.

ProLR-d reduces module density of a given placement while keeping wirelength low. The changes in wirelength and density are nearly monotonic. Unlike ProLR-w, ProLR-d consists of up to 15 LR iterations, and bin-specific θ are reset to θ_d^k of each ProLR-d invocation. ProLR-d initially rejects abrupt moves that greatly impact wirelength, and increasing θ_d^k progressively puts a greater emphasis on spreading over multiple invocations. In contrast to ProLR-w, EBB^- is applied to bins with below-target utilization, attracting modules to sparse bins. We repeat ProLR-d up to 12 times until ABU_{10} stabilizes.

Refinement. When a cluster is broken down, constituent modules are placed side by side. The placement is refined by ProLR.[5] Note in Figures 5.1 and 5.2 that during disruptions, wirelength increases sharply and density decreases. Therefore, we schedule ProLR-d before the disruption and ProLR-w after the disruption. Figure 5.4 shows that this schedule smoothens disruptions in both objectives.

Hand-off to detailed placement. Preprocessing lower-bound placements by ProLR gives

[5] Unclustering is followed by *interpolation* in [25,31] to improve ordering, but ProLR explicitly optimizes HPWL and module density.

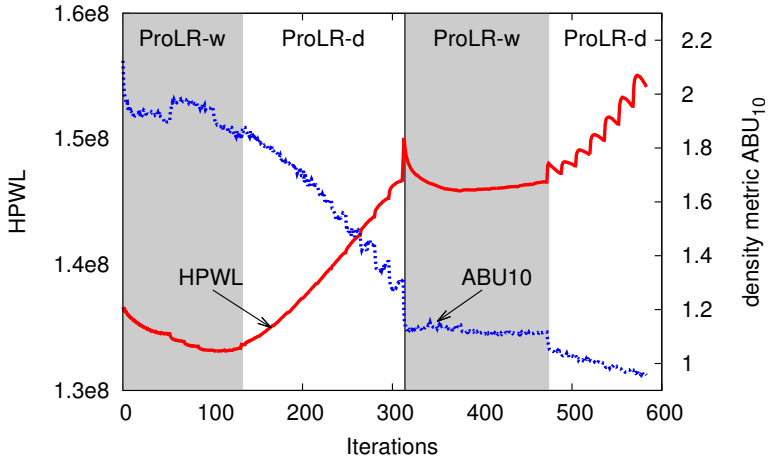


Figure 5.4: Progressions of wirelength and the density metric ABU_{10} over ProLR iterations (BIGBLUE2). Unclustering is marked with a vertical line. ProLR alternates ProLR-w (shaded) and ProLR-d phases.

better tradeoffs between wirelength and density than passing either upper-bound or lower-bound placements to detailed placement algorithms as in original SimPL [86].

5.6 Placing Macro blocks

In placers based on nonconvex optimization, the handling of pre-placed macro blocks requires dedicated techniques (sigmoid functions, level smoothing, etc). In MAPLE, the handling of pre-placed macro blocks is inherited from the SimPL algorithm [86] and LR. To handle movable macros, we extend *lookahead legalization (LAL)* of SimPL, and call the resulting step *E-LAL*. With E-LAL, upper-bound placements are generated in two steps: macro positions are determined first, followed by standard-cell placement [110]. As in original SimPL, roughly legalized placements generated by E-LAL produce fixed pseudopins for subsequent quadratic optimization. Movable macros are legalized by a variant of the *cell shifting* algorithm in FastPlace2 [139]. Our variant uses larger regular bins at 6 times the row height, and employs a 3×3 Laplacian [140] to smoothen bin utiliza-

tion. A broader view of utilization allows E-LAL to move macros further than FastPlace-Global can and find an almost-legal placement. In the early top-level placement iterations, MAPLE simultaneously places movable macros and standard cells. Upon stabilization (when the gap between the upper- and lower-bounds reduces below 50% from the gap at the 10^{th} iteration), we fix only movable macros with heights that are greater than 2 times the row height. Further iterations optimize locations of standard and double-height cells (Figure 5.5). Recent macro placement literature [32, 33] points out that naive force-directed methods do not reliably find overlap-free placements and that a poor macro placement may cause large overlaps and substantial disruption when removing those overlaps. To address this problem, unlike other force-directed placers, MAPLE fixes macro positions from the upper-bound placement, which tend to have little overlap among macros (Figure 5.5). Local refinement (LR) moves double-height and standard cells. For double-height cells, bin-specific θ_b and the utilization weights are averaged over all relevant bins. Following the contest protocol, flipping and rotation of macro blocks were disallowed in this chapter. While macro placement [32, 33, 110] is not a *primary* focus of this chapter, our techniques produce competitive results on ISPD 2006 benchmarks. Ongoing work indicates that our algorithms for mixed-size placement can be improved further.

5.7 Empirical Validation

The MAPLE algorithm is implemented in C/C++ within an industry infrastructure for placement optimization, including a variant of FastPlace-DP [112] for final legalization and detailed placement. We compared MAPLE to other state-of-the-art academic and industry placers on the ISPD 2005 and ISPD 2006 placement contest benchmark suites. For

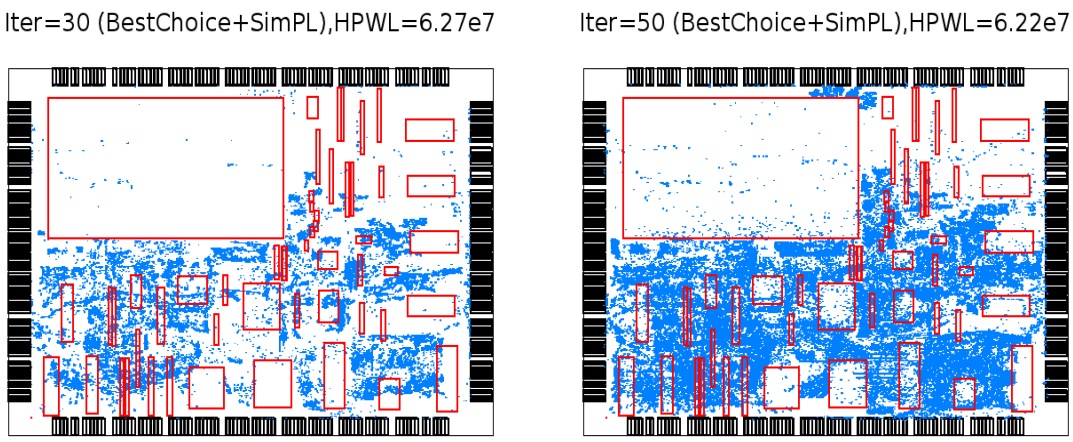


Figure 5.5: Macro placement on NEWBLUE1. (left) Macros are fixed at top-level placement iteration 30. (right) Further iterations optimize cell locations.

placers available to us, benchmark runs were performed on an Intel Core i7 860 Linux workstation running at 2.8GHz with 8GB RAM, using only one CPU core. For other placers (marked with asterisks), results were quoted from respective publications. To ensure the reproducibility of our empirical results, Formula 9 reports specific constants used in our experiments. All benchmarks were placed with identical parameter settings. HPWL of solutions produced by each placer was computed by the GSRC Bookshelf Evaluator [1].

5.7.1 ProLR versus ILR

Figure 5.6 illustrates the use of ProLR and ILR in MAPLE through snapshots of placements at different phases of Algorithm 3, starting with identical placements at Phase1. The use of ILR in Phase2a relocates many cells over great distances across fixed macros, as seen in the upper left regions of ILR plots on the left. These moves decrease maximal density, but change the placement abruptly and increase HPWL. After Phase2b, the difference in HPWL between ILR and ProLR decreases, but ILR results remain inferior. One

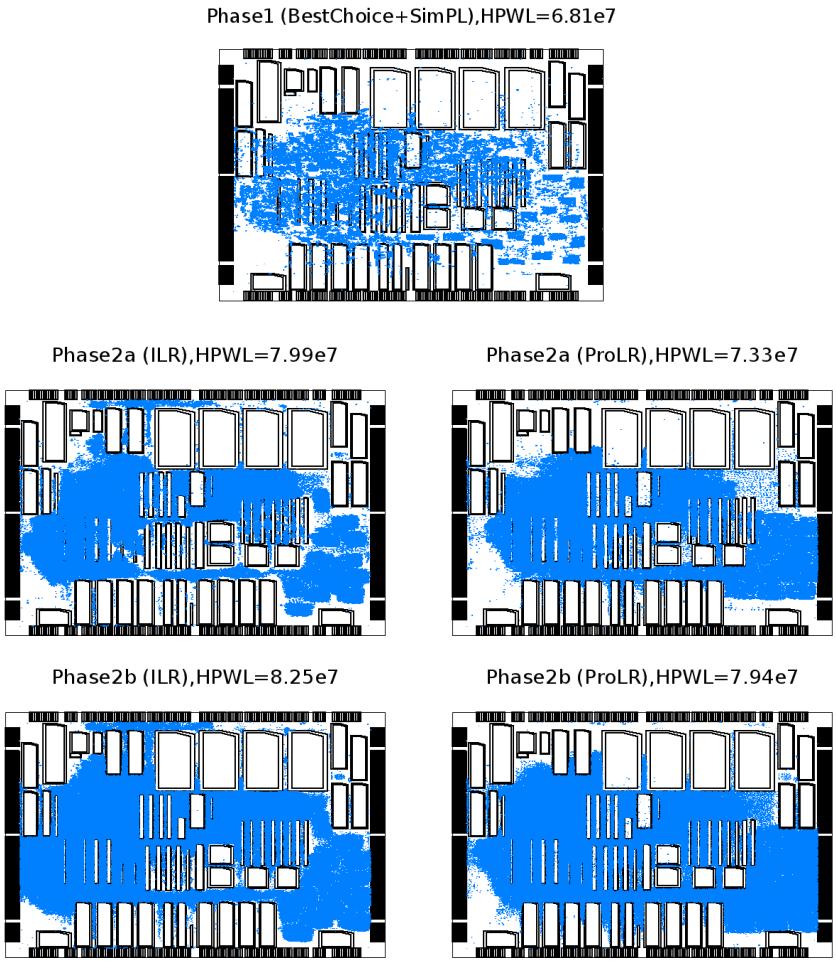


Figure 5.6: Snapshots of global placement (ADAPTEC1) after each phase of Algorithm 3 for MAPLE with ILR (left) and MAPLE with ProLR (right). Phase1 is top-level placement (BestChoice+SimPL). Phase2a and Phase2b perform LR placement of the coarsened and flat netlist, respectively.

can also see that ILR placements on the left are more clustered than the ProLR placements on the right *and* deviate more from the top-level placements. Table 5.1 compares MAPLE with ProLR to MAPLE with ILR on ISPD 2005 benchmarks in terms of final HPWL. The results confirm the superiority of ProLR. On the two largest benchmarks — BIGBLUE3 and BIGBLUE4, ProLR was on average, 1.5 times slower than ILR.

5.7.2 Comparisons on ISPD 2005 testcases

As shown in Table 5.3, MAPLE found placements with the lowest HPWL for seven out of eight circuits in the ISPD 2005 benchmarks (no parameter tuning to specific benchmarks was employed). On average, MAPLE improves wirelength by 9.50%, 6.24%, 6.53%, 7.10%, 8.06%, 4.72%, 2.73% and 2.09% versus APlace2 [79], NTUPlace3 (V7.05.30) [36], FastPlace3 [140], Kraftwerk2 [134], mFAR [66], mPL6 [26], SimPL [86] and RQL [141], respectively.

Table 5.2 compares the runtime of MAPLE with mPL6, APlace2, NTUPlace3, FastPlace3 and SimPL. On average, MAPLE is 1.13 times, 2.68 times faster than mPL6, APlace2, and 2.32 times, 6.25 times, and 7.14 times slower than NTUPlace3, FastPlace3 and SimPL, resp. On BIGBLUE4, **top-level placement iterations** consume 26.3% of total runtime: 64.1% is in CG, and 18.3% in building sparse matrices for CG. **ProLR iterations** consume 65.4% total time split almost evenly between ProLR-w and ProLR-d. **Best-choice clustering and unclustering** consume 0.2% of total runtime. **Detailed placement** takes 5.5%.

Ckts	MAPLE w/ ILR	MAPLE w/ ProLR	Improv.
AD1	77.41	76.36	1.37%
AD2	89.07	86.95	2.38%
AD3	210.13	209.78	0.17%
AD4	190.07	179.91	5.35%
BB1	95.25	93.74	1.59%
BB2	149.84	144.55	3.53%
BB3	345.20	323.05	6.42%
BB4	792.20	775.71	2.08%
Avg	**1.03**×	**1.00**×	**2.86%**

Table 5.1: HPWL (×10e6) produced by ProLR and ILR on ISPD 2005 benchmarks ADAPTEC (AD) and BIGBLUE (BB).

5.7.3 Runtime considerations

As MAPLE is currently slower than some of its competitors, we note that industry implementations like ours tend to be handicapped (versus standalone academic implementations) by the use of a multipurpose design database. Because such a database stores information unnecessary to placement, the decreased cache locality increases runtime. Other relevant legacy infrastructures in our database include netlist-query support for accurate timing analysis and physical synthesis. In contrast to academic placers, our implementation can work with a netlist that is dynamically changed during physical synthesis.

Unlike the original SimPL, our implementation does not use SSE instructions and is almost twice as slow (so far, we focused on solution quality and not runtime). Also, ProLR should parallelize well on multicore CPUs. Another consideration deals with the role of placement in physical synthesis, where it is invoked several times [10]. Fast execution is particularly important for early runs that estimate interconnect before netlist optimization. The top-level placement step from MAPLE produces good estimates because the final placement result does not look very different (Figure 5.6). Top-level placement consumes only $25 - 30\%$ of MAPLE runtime and can be accelerated as outlined above. As timing

Ckts	AP2	NTU3	MPL6	FP3	SIMPL	MAPLE
AD1	46.29	7.92	21.45	2.36	2.48	17.48
AD2	65.49	7.28	21.87	3.58	3.46	24.30
AD3	144.27	14.98	67.14	7.56	6.43	47.34
AD4	158.30	15.47	57.70	6.69	5.44	44.32
BB1	56.68	12.67	24.56	3.67	3.53	24.31
BB2	110.96	25.18	65.44	6.51	6.36	43.96
BB3	233.70	49.70	88.87	19.85	13.25	94.36
BB4	516.37	109.82	199.74	32.27	29.50	214.86
Avg	2.68×	0.43×	1.13×	0.16×	0.14×	1.00×

Table 5.2: Runtime comparison (minutes) on ISPD 2005 benchmarks for APlace2 (AP2), NTUPlace3 (NTU3), mPL6, FastPlace3 (FP3), SimPL and MAPLE.

analysis and optimizations dominate the runtime of physical synthesis, greater effort in placement can be justified by improved results.

Runtime can sometimes be reduced by *deeper clustering* (more levels). To estimate its potential impact in MAPLE, we note that top-level placement takes 26% and ProLR takes 65% of MAPLE runtime on BIGBLUE4 (195.52 min. / 91% total). ProLR runtime is split 1:2 between the coarse and flat netlists. For *three levels* of clustering, top-level placement will take 13%, and ProLR will take 11% + 22% + 43% = 76% runtime. The total (191.23 min. / 89%) is only a 2% reduction versus *two levels*.

Benchmarks	AP2 [79]	NTU3 [36]	FP3 [140]	Kw2* [134]	mFAR* [107]	mPL6 [26]	SimPL [86]	RQL* [141]	MAPLE
ADAPTEC1	78.35	81.82	78.66	82.43	82.50	77.93	78.58	77.82	**76.36**
ADAPTEC2	95.70	88.79	94.06	92.85	92.79	92.04	91.24	88.51	**86.95**
ADAPTEC3	218.52	214.83	214.13	227.22	217.56	214.16	**208.90**	210.96	209.78
ADAPTEC4	209.28	195.93	197.50	199.43	197.90	193.89	185.39	188.86	**179.91**
BIGBLUE1	100.02	98.41	96.67	97.67	98.80	96.80	97.54	94.98	**93.74**
BIGBLUE2	153.75	151.55	155.74	154.74	160.40	152.34	145.28	150.03	**144.55**
BIGBLUE3	411.59	360.66	365.16	343.32	368.70	344.10	340.24	323.09	**323.05**
BIGBLUE4	871.29	866.43	836.20	852.40	865.40	829.44	801.35	797.66	**775.71**
Geomean	**1.10**×	**1.07**×	**1.07**×	**1.08**×	**1.09**×	**1.05**×	**1.03**×	**1.02**×	**1.00**×

Table 5.3: Legal HPWL (×10e6) comparison on the ISPD 2005 benchmark suite. The previous best wirelengths are marked with gray. The placers marked by asterisks were unavailable to us in binary, and we reproduce HPWL from respective publications. Kw2=Kraftwerk2.

5.7.4 Comparisons on ISPD 2006 testcases

We compared MAPLE to other state-of-the-art academic and industry placers on the ISPD 2006 benchmark suite. Table 5.4 reports scaled HPWL and overflow penalty for several placers. Following the contest protocol, scaled HPWL is calculated as $HPWL \cdot (1 + 0.01 \cdot overflow\_penalty)$. On average, MAPLE achieved 11.28%, 5.59%, 13.58%, 6.63%, 11.57%, 4.37%, 3.13% scaled HPWL improvements versus APlace3 [107], NTU-

Benchmarks (Υ_{target})	AP3* [107]	NTU3 [36]	FP3 [140]	Kw2* [134]	mFAR* [107]	mPL6 [26]	RQL* [141]	MAPLE
ADAPTEC5 (0.5)	520.97 (15.9)	430.73 (12.2)	541.22 (36.5)	449.84 (3.69)	476.28 (6.21)	431.27 (1.09)	443.28 (9.25)	**407.33 (4.76)**
NEWBLUE1 (0.8)	73.31 (0.14)	**62.39 (0.76)**	76.56 (1.02)	65.95 (0.05)	77.54 (0.23)	68.08 (0.14)	64.43 (0.34)	69.25 (1.05)
NEWBLUE2 (0.9)	198.24 (0.42)	211.77 (3.21)	240.56 (1.97)	206.53 (1.28)	212.90 (0.59)	201.85 (1.52)	199.60 (1.45)	**191.66 (1.01)**
NEWBLUE3 (0.8)	273.64 (0.00)	280.19 (0.01)	301.72 (0.78)	279.58 (0.38)	303.91 (0.11)	284.11 (0.59)	269.33 (0.07)	**268.07 (0.77)**
NEWBLUE4 (0.5)	384.12 (1.74)	302.25 (9.22)	306.07 (7.74)	309.44 (1.71)	324.40 (5.42)	300.58 (1.63)	308.75 (15.2)	**282.49 (5.86)**
NEWBLUE5 (0.5)	613.86 (12.5)	547.20 (20.82)	633.72 (28.31)	563.15 (2.69)	601.27 (5.92)	537.14 (1.42)	537.49 (13.6)	**515.04 (4.05)**
NEWBLUE6 (0.8)	522.73 (0.03)	518.25 (6.08)	531.56 (1.26)	537.59 (1.70)	535.96 (1.63)	522.54 (1.40)	515.69 (4.33)	**494.82 (1.08)**
NEWBLUE7 (0.8)	1098.9 (0.06)	1114.2 (5.19)	1116.7 (1.33)	1162.1 (3.15)	1153.8 (1.58)	1084.4 (1.14)	1057.8 (2.57)	**1032.6 (1.70)**
Geomean	**1.13 × (0.32)**	**1.04 × (2.55)**	**1.16 × (3.47)**	**1.07 × (1.09)**	**1.13 × (1.29)**	**1.06 × (1.22)**	**1.03 × (2.30)**	**1.00 × (1.90)**

Table 5.4: Comparison of scaled HPWL ($\times$10e6) which includes overflow penalty w.r.t the given target utilization on the ISPD 2006 benchmark suite. Overflow penalty values computed by the contest script are reported in parentheses. The placers marked by asterisks were unavailable to us in binary, and we reproduce results from respective publications. This hinders runtime comparisons.

Place3 (V7.05.30) [36], FastPlace3 [140], Kraftwerk2 [134], mFAR [66], mPL6 [26], and RQL [141], respectively. MAPLE obtains the best scaled HPWL results on seven out of eight circuits. Furthermore, compared to the other two best-performing placers on the benchmarks — RQL and NTUPlace3, MAPLE achieves lower overflow penalty on average. Thus, MAPLE not only reduces the wirelength but also avoids highly concentrated placements. Recall that the original implementation of SimPL [86] does not support density constraints of ISPD 2006 benchmarks and does not perform mixed-size placement.

5.8 Summary and Conclusions

The significance of large-scale placement in IC physical design is well-documented in recent literature [10] and is continuing to grow with the amount of on-chip random logic and current trends in interconnect scaling. Placement algorithms in the industry and academia were initially developed with the HPWL objective in mind [107] and later extended [10] to account for other objectives and concerns [57, 64, 85]. Despite known pitfalls, the HPWL objective appears to be a good performance predictor for various extensions of core placement algorithms. Focusing on the HPWL objective and module density, our research (i) contributes the discovery of essential deficiencies in prior techniques and (ii) advances the state of the art by developing algorithms that improve the quality of benchmark layouts beyond all published results.

PART III

Broadening the Scope of Placement Optimization

CHAPTER VI

ComPLx: A Competitive Primal-dual Lagrange Optimization for Global Placement with Extensions to Mixed-size and Timing-driven Placement

We develop a projected primal-dual Lagrange optimization for global placement, that can be instantiated with a variety of interconnect models. It decomposes the original non-convex problem into "more convex" sub-problems. It generalizes and extends our SimPL algorithm introduced in Chapter III, the SimPLR algorithm from Chapter IV and the related Ripple algorithm [57] derived by our colleagues from SimPL. Empirically, ComPLx demonstrates strong results in runtime and performance on ISPD 2005 and 2006 benchmark suites.

6.1 Introduction

The success of global placement determines all aspects of modern IC layout and physical synthesis [10] because it controls the amount of interconnect, which increasingly dominates on-chip resources and circuit performance [80]. However, the diverse algorithmic challenges posed by global placement and its complexity continue to surprise researchers [107]. Current algorithms still lag behind manual layout on circuits with struc-

tured components [149], do not always scale to extremely large circuits and are inconsistent in handling of objective functions and various constraints. Analysis and comparisons of placement algorithms have been mostly empirical [107], with little formal justification and convergence analysis.

A recent approach to global placement promises to support a variety of discrete and continuous constraints and was extended to handle routability-driven placement. Represented by the SimPL [86] and SimPLR [85] algorithms, this approach consistently outperforms previous state of the art in speed and solution quality, is amenable to thread-level and instruction-level parallelism, requires only a modest amount of code, and was successfully re-implemented by independent researchers [14,57]. The SimPL algorithm saw rapid adoption in both academia and industry, including power-aware placement with integrated clock-network synthesis in [94] and multilevel optimization in [87]. Recently, it was also extended to thermal-aware placement in [14] and datapath-aware placement in [151, 152]. However, a convincing mathematical foundation for this empirical success was lacking. While the SimPL approach is based on quadratic placement, the significance of this connection has remained unclear *vis-à-vis* techniques based on the log-sum-exp interconnect model [126].

Our contributions in this chapter can be summarized as follows

- A projected primal-dual Lagrange optimization (ComPLx) for global placement compatible with a variety of interconnect models, including linearized quadratic, log-sum-exp, etc.

- Convergence analysis and ensuing enhancements.

- Casting existing algorithms SimPL [86], SimPLR [85] and Ripple [57] as special cases of ComPLx. In particular, ComPLx inherits their competitiveness and lends them mathematical substantiation.

- Algorithmic extensions for mixed-size, timing- and power-driven placement.

- Empirical validation of the theoretical framework underlying ComPLx. On ISPD 2005 benchmarks, ComPLx is 10% faster than FastPlace [140] (including detailed placement runtime). It outperforms SimPL and RQL (the best-published placers) by 1%. On ISPD 2006 benchmarks, ComPLx outperforms the leading placer RQL [141] by 1% in terms of scaled HPWL while running 2.5$\times$ faster.

In the remainder of this chapter, Section 6.2 introduces our primal-dual Lagrangian relaxation ComPLx, whose convergence is discussed in Section 6.3. Section 6.4 points out that the SimPL, SimPLR and Ripple algorithms are special cases of ComPLx. It also discusses other related prior work. Section 6.5 outlines extensions to mixed-size, timing and power-driven placement. Section 6.6 presents empirical studies that evaluate ComPLx and its extensions, and compare ComPLx to prior art. Summary and conclusions are given in Section 6.7.

6.2 A Primal-Dual Lagrange Method

We propose a general method for handling constraints in global placement with a variety of possible interconnect models, and show how to decompose the original nonconvex problem into "more convex" sub-problems.

A Lagrangian relaxation of global placement can be constructed if constraints are spec-

ified as *equalities* $\Pi(\vec{x}, \vec{y}) = 0$. Since supply-demand *inequalities* are usually given instead, the more general Karush-Kuhn-Tucker conditions may at first seem more relevant.[1] However, working with supply-demand inequalities directly is difficult because they are specified algorithmically, not as closed-form expressions in $(\vec{x}, \vec{y})$. Without derivatives, one resorts to *subgradient optimization* [15], while the nature of the constraints calls for approximation. Placement techniques based on nonconvex optimization [26, 36, 79] fit demand distribution to smooth functions using *kernel-density estimation*, and this facilitates gradient estimation. Each such step is laborious, and many steps may be required because, after moving in the gradient direction, one may need to "make turns" (as illustrated by moving around a rectangular obstacle). The reliance on local subgradient information in [36, 79] is common in analytical placement and with possible exceptions of Kraftwerk [134] and mPL6 [26] which estimate subgradients by solving second-order linear elliptic PDEs with global supply-demand information. Solutions of these PDEs can be written as convolutions of the density function with a fixed Green's function $G(s, t)$ (dependent on boundary conditions), which sometimes vanishes away from $s = t$. Further, local subgradient computations leave undefined the tradeoff between demand-distribution subgradients and the gradients of the objective function. This *force modulation* problem was articulated in [141], but addressed there with *ad hoc* thresholding.

In contrast to other methods, our subgradients point to a closest $\mathcal{C}$-feasible solution, and their magnitude is modulated by respective distance. Thus, we define $\Pi_{\mathcal{C}}(\vec{x}, \vec{y})$ as the

[1] Inequalities can also be converted into equations by adding slack variables, but we avoid this common technique, to limit computational complexity.

L_1-distance from $(\vec{x}, \vec{y})$ to a closest $\mathcal{C}$-feasible solution.

$$\Pi_{\mathcal{C}}(\vec{x}, \vec{y}) = \min_{(\vec{x}_*, \vec{y}_*) \in \mathcal{C}} ||(\vec{x}, \vec{y}) - (\vec{x}_*, \vec{y}_*)||_1 \tag{6.1}$$

$$= \min_{(\vec{x}_*, \vec{y}_*)} \left(||\vec{x} - \vec{x}_*||_1 + ||\vec{y} - \vec{y}_*||_1 \right) \tag{6.2}$$

Clearly, $\Pi_{\mathcal{C}}(\vec{x}, \vec{y}) = 0 \;\Leftrightarrow\; (\vec{x}, \vec{y}) \in \mathcal{C}$. Therefore, in addition to primary variables $(\vec{x}, \vec{y})$, we introduce one dual variable (multiplier) $\lambda \geq 0$, and establish the following Lagrangian

$$\mathcal{L}_{\Phi,\mathcal{C}}(\vec{x}, \vec{y}, \lambda) = \Phi(\vec{x}, \vec{y}) + \lambda \, \Pi(\vec{x}, \vec{y}) \tag{6.3}$$

We use L_1-norms so that costs and penalties are expressed in meters and can be compared. Hence, λ is dimensionless.

Primal-dual Lagrangian relaxation [6] alternates minimization over the primal variables with maximization over the dual variable(s). $\min \Phi(\vec{x}, \vec{y})$ subject to $(\vec{x}, \vec{y}) \in \mathcal{C}$ can be found by *sequential unconstrained optimization*

$$\max_{\lambda} \min_{(\vec{x}, \vec{y})} \mathcal{L}_{\Phi,\mathcal{C}}(\vec{x}, \vec{y}, \lambda) \tag{6.4}$$

Starting with $\lambda_0 = 0$, the first primal iterate is produced by minimization of $\Phi(\vec{x}, \vec{y})$ (using quadratic optimization or nonlinear Conjugate Gradient, depending on the function). At subsequent iterations, primal optimization must also account for the penalty term. A straightforward argument by contradiction shows that for $\lambda_k < \lambda_{k+1}$

$$\min_{(\vec{x}, \vec{y})} \mathcal{L}_{\Phi,\mathcal{C}}(\vec{x}, \vec{y}, \lambda_k) \leq \min_{(\vec{x}, \vec{y})} \mathcal{L}_{\Phi,\mathcal{C}}(\vec{x}, \vec{y}, \lambda_{k+1}) \tag{6.5}$$

As λ increases, so does the sensitivity of $\mathcal{L}_{\Phi,\mathcal{C}}$ to Π. Therefore, the minimization of $\mathcal{L}_{\Phi,\mathcal{C}}$ affects the Π term more, and this term decreases. However, since the minimized value of

$\mathcal{L}_{\Phi,\mathcal{C}}$ increases (per Formula 6.5), Φ must increase. Eventually, $(\vec{x}, \vec{y})$ become $\mathcal{C}$-feasible (or very close to), making the Lagrangian insensitive to λ and indicating that an optimum is near. The following weak duality bounds hold for any $\mathcal{C}$-feasible solution $(\vec{x}^\circ, \vec{y}^\circ)$ and any iterate $(\vec{x}, \vec{y})$ after primal optimization.

$$\Phi(\vec{x}, \vec{y}) \leq \mathcal{L}_{\Phi,\mathcal{C}}(\vec{x}, \vec{y}, \lambda) \leq \mathcal{L}_{\Phi,\mathcal{C}}(\vec{x}^\circ, \vec{y}^\circ, \lambda) = \Phi(\vec{x}^\circ, \vec{y}^\circ) \tag{6.6}$$

The first $\leq$ is due to $\lambda \geq 0$ in Formula 6.3. The second $\leq$ is due to $(\vec{x}, \vec{y})$ being argmin from Formula 6.4 and the third $=$ is due to $\Pi(\vec{x}^\circ, \vec{y}^\circ) = 0$ ($\mathcal{C}$-feasible). The second inequality is strict unless $(\vec{x}, \vec{y})$ is $\mathcal{C}$-infeasible, hence $\Phi(\vec{x}, \vec{y}) < \Phi(\vec{x}^\circ, \vec{y}^\circ)$. The *duality gap* is

$$\Delta_\Phi = \Phi(\vec{x}^\circ, \vec{y}^\circ) - \Phi(\vec{x}, \vec{y}) \tag{6.7}$$

minimized over best available *primal feasible* $(\vec{x}^\circ, \vec{y}^\circ)$ and $(\vec{x}, \vec{y})$ at a given point during optimization.

Approximating the penalty term allows us to replace the nonconvex Lagrangian by a convex one. Here we use the *feasibility projection*

$$P_\mathcal{C}(\vec{x}, \vec{y}) = \text{argmin}_{(\vec{x}_*, \vec{y}_*) \in \mathcal{C}} ||(\vec{x}, \vec{y}) - (\vec{x}_*, \vec{y}_*)||_1 \tag{6.8}$$

that finds a closest $\mathcal{C}$-feasible approximation (performs *pseudo-legalization*) of $(\vec{x}, \vec{y})$.[2] Since $P_\mathcal{C}(\vec{x}', \vec{y}')$ is $\mathcal{C}$-feasible, $\Phi(\vec{x}, \vec{y}) \leq \Phi\big(P_\mathcal{C}(\vec{x}', \vec{y}')\big)$ by Inequalities 6.6. Given that Φ is continuous, the convergence $||(\vec{x}, \vec{y}) - P_\mathcal{C}(\vec{x}, \vec{y})||_1 \to 0$ would necessitate $\Phi\big(P_\mathcal{C}(\vec{x}, \vec{y})\big) - \Phi(\vec{x}, \vec{y}) \to 0$. Hence, $\Phi\big(P_\mathcal{C}(\vec{x}, \vec{y})\big)$ must generally decrease, providing upper bounds on final placement cost.

[2]One can additionally require breaking ties toward smaller values of Φ (or even some tradeoff with Φ), but this does not seem necessary for practical success (Section 6.6).

After finding $\mathcal{C}$-feasible *anchor locations* $(\vec{x}^\circ, \vec{y}^\circ) = P_\mathcal{C}(\vec{x}, \vec{y})$, we establish the simplified Lagrangian

$$\mathcal{L}^\circ_\Phi(\vec{x}, \vec{y}, \lambda) = \Phi(\vec{x}, \vec{y}) + \lambda||(\vec{x}, \vec{y}) - (\vec{x}^\circ, \vec{y}^\circ)||_1 \tag{6.9}$$

To minimize it with respect to fixed $(\vec{x}^\circ, \vec{y}^\circ)$ and λ, the L_1-term can be approximated by the same type of function as Φ (see Section 6.4). Thus, for quadratic Φ, the optimality condition $\nabla\mathcal{L}^\circ_\Phi(\vec{x}, \vec{y}, \lambda) = 0$ turns into a system of linear equations. For other functional forms, such as the log-sum-exp expressions, one can minimize $\mathcal{L}^\circ_\Phi(\vec{x}, \vec{y}, \lambda)$ using the nonlinear Conjugate Gradient method or other known alternatives.[3] In addition to being (strictly) convex, $\mathcal{L}^\circ_\Phi(\vec{x}, \vec{y}, \lambda)$ is usually separable into its x and y components which can be optimized independently. One can verify Inequalities 6.5 and 6.6 for $\mathcal{L}^\circ_\Phi(\vec{x}, \vec{y}, \lambda)$ subject to $(\vec{x}^\circ, \vec{y}^\circ) = P_\mathcal{C}(\vec{x}, \vec{y})$.

The ComPLx framework re-solves $\nabla\mathcal{L}^\circ_\Phi(\vec{x}, \vec{y}, \lambda) = 0$ and $P_\mathcal{C}(\vec{x}, \vec{y}) = 0$ until convergence. The result of global placement can be read from the last iterate $(\vec{x}, \vec{y})$ or the last $\mathcal{C}$-feasible iterate $(\vec{x}^\circ, \vec{y}^\circ)$ as discussed in Section 6.3.

Feasibility projection for density constraints. The notion of a feasibility projection is related to lookahead legalization (LAL) in the SimPL placer [86] (more details in Section 6.4). To handle density constraints, ComPLx uses a similar algorithm, which we here restructure to expose the convexity of underlying optimization.

Whereas LAL was defined recursively in [86, Section 4], the top-level structure of $P_\mathcal{C}$ in our description is that of alternating horizontal and vertical *spreading* passes. Each pass operates over a slicing floorplan, which gets refined between the passes. Specifically,

[3] Techniques such as Newton's method that approximate the objective f by quadratic functions based on Hessian(f) essentially perform sequential quadratic optimization.

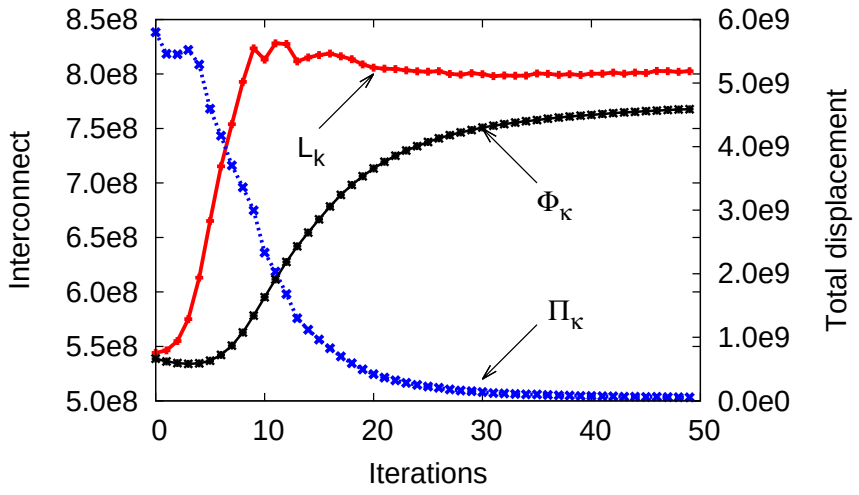


Figure 6.1: Progressions of $\mathcal{L}$ (the total Lagrangian), Φ (netlist interconnect), and Π (L_1-distance to legal) over ComPLx iterations on BIGBLUE4. $\mathcal{L}$ increases steeply in the early placement iterations, as λ increases. Π decreases while Φ gradually increases.

spreading occurs only inside the rooms of the floorplan. For example, at the very first iteration, there is only one room, and one-dimensional spreading evens out the density. To formalize the problem solved by one-dimensional spreading from [86, Section 4], we note that relative placements are preserved. This justifies a change of variables: initial cell locations x_i (or y_i) are sorted, and the new variables $\delta_i \geq 0$ represent distances between neighboring x (y) locations, subject to $\Sigma_i \delta_i \leq W_x$ (or W_y) for a $W_x \times W_y$ floorplan room (a convex constraint). This linear change of variables preserves the convexity of the optimization objective (L_1-distance from given locations). The density constraint requires that for (some m and) all k, $\Sigma_{i=k}^{k+m} \delta_i$ is sufficiently large (based on cell sizes). This constraint

$$\min_k \{\Sigma_{i=k}^{k+m} \delta_i - (1/\sqrt{\gamma}) \Sigma_{i=k}^{k+m} \text{width}(\text{cell}_i)\} \geq 0$$

is convex since the minimum of downward convex (linear) functions is also downward convex (the $\geq$ sign is important).

As pointed out in [86, Section 4], after one-dimensional spreading, the median location should divide cell area evenly. Since this equalizes average densities on both sides, this new median location indicates a fixed point of the spreading transform. Hence, the walls of the slicing floorplan built by alternating one-dimensional spreading steps represent fixed lines. As slicing floorplan is gradually refined, the displacement affected by later steps of $P_{\mathcal{C}}$ rapidly decreases.

6.3 Convergence analysis

It is sufficient for $P_{\mathcal{C}}$ to find a $\mathcal{C}$-feasible solution that is *reasonably close*, rather than closest, to a given $(\vec{x}, \vec{y})$. Such *approximate projected subgradient* methods are relatively recent in the operations-research literature [89, Section 1] but are proven to converge as long as $P_{\mathcal{C}}$ does not increase the distance to the set $\mathcal{C}$ and typically reduces it during iterations [16, Sections 2 and 3]. In particular, $P_{\mathcal{C}}$ should return its input when the input is $\mathcal{C}$-feasible. Convergence can be improved if $P_{\mathcal{C}}$ exhibits reasonable *fidelity* with respect to the exact feasibility projection, and is *self-consistent*

$$||(\vec{x}, \vec{y}) - P_{\mathcal{C}}(\vec{x}, \vec{y})||_1 > ||(\vec{x}', \vec{y}') - P_{\mathcal{C}}(\vec{x}, \vec{y})||_1 \quad \Rightarrow \tag{6.10}$$

$$||(\vec{x}, \vec{y}) - P_{\mathcal{C}}(\vec{x}', \vec{y}')||_1 > ||(\vec{x}', \vec{y}') - P_{\mathcal{C}}(\vec{x}', \vec{y}')||_1 \tag{6.11}$$

In other words, if $(\vec{x}', \vec{y}')$ is closer to $P_{\mathcal{C}}(\vec{x}, \vec{y})$ than $(\vec{x}, \vec{y})$, then it should also be closer to $P_{\mathcal{C}}(\vec{x}', \vec{y}')$. The ComPLx implementation of $P_{\mathcal{C}}$ reviewed in Section 6.4 handles both standard cells and macros. It is self-consistent through almost all iterations, as we shown empirically in Section 6.6. Figure 6.1 illustrates changes in $\mathcal{L}_k$, Φ_k, and Π_k over ComPLx iterations on BIGBLUE4. The same trends show on all other benchmarks, validating the

discussion in Section 6.2.

Global placement iterations stop when a $\mathcal{C}$-feasible value is reached, which must happen when λ exceeds its optimal value. But the resulting solution may be far from optimal. To avoid this, we propose to improve the efficiency of the first few iterations, since the first iterates are crucial to the overall success (given that we are solving a nonconvex problem overall). The earliest nonzero value of λ must be sufficiently small so that $\Phi(\vec{x}, \vec{y}) \gg \lambda\Pi(\vec{x}, \vec{y})$, to make sure that $\mathcal{L}_{\Phi,\mathcal{C}}(\vec{x}, \vec{y}, \lambda)$ is dominated by the convex *cost* term rather than the penalty term. Hence, we initially select $\lambda_1 = \Phi/100\Pi$. This calculation is supported by the fact that Π and Φ are expressed in the same units (meters). To *avoid* premature progress, a maximum increase in λ can be imposed, say 100% per iteration.

$$\lambda_{k+1} = \min\{2\lambda_k, \lambda_k + (\Pi_{k+1}/\Pi_k)h\} \tag{6.12}$$

where h is a scaling constant. λ increases proportionally to Π changes to ensure that Π decreases by a sufficient amount, as Φ increases. Considering the number of iterations until λ reaches its optimal value, there is no explicit dependency on the number of variables. In practice, the maximal λ values and the iteration count do not grow with the size of the problem instance as shown in Section 6.6.

Convergence criteria can be defined in terms of $r(\vec{x}, \vec{y}) = ||(\vec{x}, \vec{y}) - P_{\mathcal{C}}(\vec{x}, \vec{y})||_1$, rather than $\mathcal{L}$ — when the placement is close to $\mathcal{C}$-feasible, a detailed placer can produce optimized site-aligned legal locations. Given that pseudo-legalization $(\vec{x}^\circ, \vec{y}^\circ) = P_{\mathcal{C}}(\vec{x}, \vec{y})$ is performed at every iteration, one can run detailed placement on $(\vec{x}^\circ, \vec{y}^\circ)$ rather than on $(\vec{x}, \vec{y})$. This would allow an even more aggressive convergence criterion in terms of the

duality gap $\Delta_\Phi = \Phi(\vec{x}^\circ, \vec{y}^\circ) - \Phi(\vec{x}, \vec{y})$.[4] To substantiate this idea, we observe that performing detailed placement on a feasible solution $(\vec{x}^\circ, \vec{y}^\circ)$ should *not* increase costs (rather the opposite), whereas performing detailed placement on $(\vec{x}, \vec{y})$ is likely to (as observed in practice). This observation upper-bounds the difference in final costs between these two scenarios by Δ_Φ.

6.4 Comparisons to Previous Work

We now point out that the SimPL (Section III), SimPLR (Section IV) and Ripple [57] algorithms are special cases of the proposed primal-dual Lagrangian relaxation. They implement Φ as a quadratic approximation Φ_Q of HPWL, adjusted at every iteration through the linearized Bound2Bound net model [134]. Linearization is also applied to represent the L_1-norm in the penalty term Π. To model this term, each movable object is connected to its anchor location by a *pseudonet*, contributing $w_i(x_i - x_i^\circ)^2$ to the overall objective (and a similar y-term), where $w_i = \frac{\lambda}{|x_i - x_i^\circ| + \varepsilon}$ is based on the last iterate. $\varepsilon > 0$ is used to bound the denominator away from zero and make the objective function strictly convex. In SimPL and SimPLR, ε is calculated as 1.5 times row height.[5] This matches Formula 6.9 if the L_1-distance term is approximated by a linearized quadratic function (Section 2.1). SimPL, SimPLR and Ripple maintain a lower and an upper-bound placement at each iteration, and these placement satisfy conditions in Formula 6.6 as seen in [86, Figure 6], [85, Figure 4].

The SimPL (Section III), SimPLR (Section IV) and Ripple [57] algorithms differ in how they define and implement the feasibility projection $P_\mathcal{C}$. In practice, to identify over-

[4] As Φ is Lipschitz, $r(\vec{x}, \vec{y}) \to 0$ implies $\Delta_\Phi \to 0$.

[5] In [132], a lower bound on the distance between two modules is defined as the average module width.

filled bins with respect to a *target utilization/density limit* $0 < \gamma \leq 1$ [86, Section 4], a uniform grid is superimposed over the entire layout. Then the feasibility projection seeks to satisfy the given target utilization/density limit within each grid-cell. To this end, the SimPL $P_{\mathcal{C}}$ first localizes the changes in $(\vec{x}, \vec{y})$ to the smallest rectangular grid-cell sub-arrays that satisfy a given target utilization/density limit, and then processes each region by a top-down geometric-partitioning framework. SimPL alternates (i) piecewise-linear scaling in x and y directions with (ii) spreading locations in each dimension to even out density, while preserving the relative order (determined by sorting). As a runtime tradeoff, SimPL gradually increases the accuracy of $P_{\mathcal{C}}$ as the grid-cell size decreases, and we use this feature in Section 6.6 to show that $P_{\mathcal{C}}$ does not need to be implemented precisely. SimPLR and Ripple generally follow the SimPL techniques, but are concerned with routability in addition to HPWL. Therefore, they estimate congestion after placement iterations (SimPLR calls a global router, whereas Ripple estimates congestion directly) and modify $P_{\mathcal{C}}$ to produce low-congestion placements. SimPLR preprocesses $P_{\mathcal{C}}$ by temporarily increasing the dimensions of some movable objects, so as to enhance geometric separation between them. Ripple distinguishes congestion maps for horizontal and vertical wiring, and scales minimal-sized rectangular regions differently each direction. Despite the technical differences, all these variants compute $P_{\mathcal{C}}$ by a series of convex optimizations. The use of feasibility projections is not only common between SimPL [86], SimPLR [85] and Ripple [57], but also distinguishes them from other placement algorithms. This is why SimPL, SimPLR and Ripple are particularly good at handling nonlinear, nonconvex layout constraints, such as numerous fixed obstacles present in modern SoC layouts.

Lookahead legalization (LAL) was earlier used for macro placement in PolarBear [42] and SCAMPI [110]. In both cases, the main issue was the feasibility of macro packing within a given fixed outline. Both algorithms use top-down min-cut partitioning to minimize interconnect and need to check if each geometric partition is feasible. The result of this check is binary — a positive result for both partitions allows top-down partitioning to proceed, while a negative result for one of partitions triggers backtracking or end-case processing. The locations of macro blocks in a feasible placement are typically not used directly, therefore the LAL algorithm in PolarBear does not seek to optimize any objective. In contrast, SimPL does not deal with movable macros, and the main concern for LAL in SimPL is to minimize total displacement from an initial solution, which has not been considered in PolarBear and SCAMPI. Since relevant algorithms in PolarBear and SCAMPI do not work with an initial solution, they cannot be considered *feasibility projections*. In other words, P_C in ComPLx and LAL in SimPL differ from prior work in that they pursue different goals — finding a closest feasible solution, rather than check packing feasibility. They are used in a different context (analytic placement vs. top-down min-cut placement), employ entirely different algorithms, and their results are interpreted differently (as anchors that influence the next iteration of analytic global placement). Whereas PolarBear and SCAMPI did not anticipate primal-dual Lagrange optimization in placement, the feasibility projection in ComPLx is a key element of the proposed primal-dual Lagrange formulation.

ComPLx also differs from PolarBear and SCAMPI in how it handles movable macros. Whereas prior work seeks to ensure the feasibility of macro placements at every step and

sometimes sacrifices interconnect optimization for this, ComPLx pursues a different strategy based on macro shredding — it allows for temporarily overlapping macro placements and focuses on interconnect optimization.

Related primal-dual Lagrange optimizations. Primal-dual optimization was used once in global placement in [7], where it was limited to explicit center-of-gravity (CoG) "spreading" constraints. These constraints appear in GORDIAN and GORDIAN-L algorithms [132], but not in modern placers — being convex and linear, they are insufficient to handle modern IC layouts (the 1997 implementation reported in [7] is not a full-fledged global placer). To deal with CoG constraints, [7] introduced slack variables, as is common in linearly-constrained primal-dual Lagrange optimization [53]. Instead, we deal with more general nonlinear, nonconvex constraints (such as fixed obstacles) by means of *approximate projected subgradient optimization*. Our primal-dual Lagrangian relaxation, in its basic form, requires only a single real-valued multiplier, making optimization very efficient. Unlike in [7, 53], we use the linear Conjugate Gradient (CG) method rather than the nonlinear Newton's method.

Recent *operations research* work (unrelated to EDA) by Kiwiel et al [89] discusses *approximate subgradient projected optimization*, focusing on step-size selection and convergence analysis. Unlike prior projected subgradient methods, the Lagrangian relaxation in [89] finds both primal and dual solutions. This is also a key feature of our methods. However, [89] is not solving global placement and lacks numerous domain-specific details we described. *Vice versa*, we are not using their hallmark *ballstep strategy* that bundles multiple subgradient iterations.

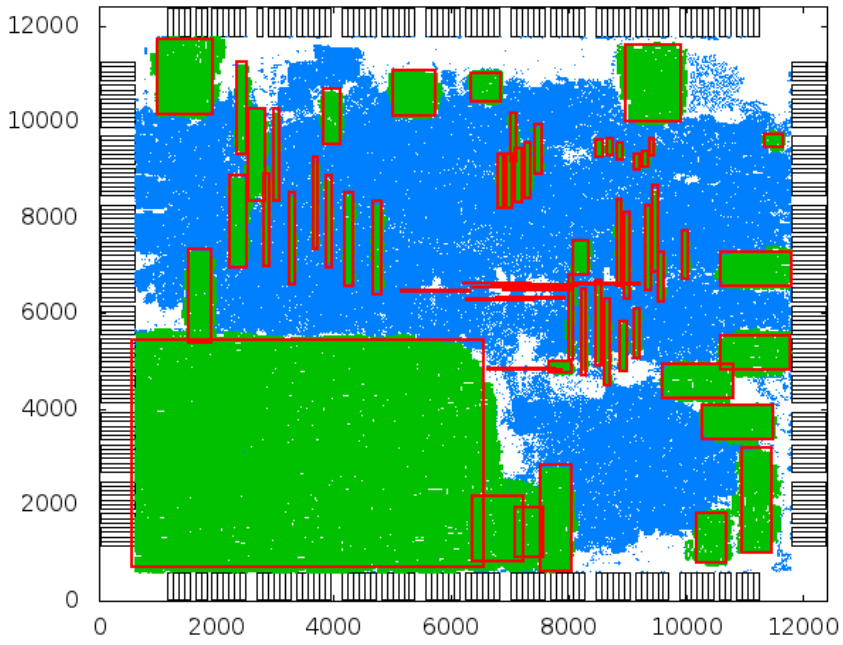


Figure 6.2: Macro shredding for feasibility projection $P_{\mathcal{C}}$ on NEWBLUE1 (an intermediate placement). Red boxes show the locations of macro cells at the centers of gravity of constituent cells (shown as green dots). Standard cells are shown as blue dots.

6.5 Algorithmic Extensions

Mixed-size placement requires careful accounting for pin offsets during quadratic optimization (since pin-offsets can be large in macros), as well as an approximate feasibility projection $P_{\mathcal{C}}$ which can handle both macros and standard cells. We have therefore revised and extended the *macro shredding technique* from [3]. Macro cells are divided into equal-sized cells (2×2 standard-cell height), but unlike prior work, ComPLx does not connect constituent cells (shreds) with fake nets and thus does not modify the linear systems it solves, limiting computational complexity increases.

The conventional $P_{\mathcal{C}}$ [86] is applied to the shreds, after which the action of $P_{\mathcal{C}}$ on the original macro is interpolated by averaging the displacement of the shreds. Given that the

conventional $P_{\mathcal{C}}$ [86] mostly preserves the relative placement of cells and is approximately locally isometric, the arrays of shreds are transformed into shapes similar to arrays, as seen in Figure 6.2. As $P_{\mathcal{C}}$ seeks to satisfy the given target utilization ($0 < \gamma < 1$), additional whitespace is inserted among constituent cells. Then the bounding box of projected locations of shreds outgrows the original macro cell, creating a halo around the macro, where other cells cannot be placed. To compensate, we multiply the widths and heights of constituent cells by $\sqrt{\gamma}$. Stabilizing macro positions early is important, as they greatly impact adjacent standard cells. To accelerate the convergence of macro cells and decrease their displacement during legalization, we extend Formulae 4 and 10 with separate, larger λ values for each macro, computed as the default λ times the ratio of the size of macro cell to the average standard-cell size. As seen in Figure 6.2, our mixed-size feasibility projection $P_{\mathcal{C}}$ may leave small overlaps between macros. Rather than force complete legalization, we let multiple global placement iterations (including $P_{\mathcal{C}}$) gradually decrease these overlaps. We observe that as $P_{\mathcal{C}}$ displaces cells and macros *less*, the changes in the shapes of shredded macros also *decrease*, and this *increases* the precision of legalization for macros during $P_{\mathcal{C}}$. Even if slight overlaps remain at the end of global placement, they can be fixed by the detailed placer without undermining the overall performance. While less sophisticated than algorithms in [32, 33, 160], our mixed-size approximate feasibility projection $P_{\mathcal{C}}$ is easy to implement and shows good results, motivating additional studies.

Timing- and power-driven placement traditionally rely on net weights computed from activity factors and timing slacks [80, Chapter 8]. Net-weighting schemes in the literature include rigorous, provably convergent methods [24]. Since our mathematical formulation

for *global placement* in Section 6.2 accounts for net weights in Φ, existing techniques [24] and their provable properties apply directly. An extension of SimPL with power-driven net weights is reported in [94]. However, we observe that *the impact of the feasibility projection and detailed placement* suggests revising the penalty term in the Lagrangian. Minimizing L_1-distance to $\mathcal{C}$ may leave some cells far from their legal positions, forcing $P_{\mathcal{C}}$ or the detailed placer to displace them. This may stretch out incident nets, which is undesirable for timing- and power-critical standard cells. Hence, in the simplified Lagrangian of Formula 6.9 we weigh the penalty term by timing/power criticality and replace

$$\lambda||(\vec{x},\vec{y}) - (\vec{x}^{\circ},\vec{y}^{\circ})||_1 \text{ by } \lambda(\vec{\gamma} \cdot |(\vec{x},\vec{y}) - (\vec{x}^{\circ},\vec{y}^{\circ})|) \tag{6.13}$$

where $|\ |$ represents the vector of pointwise distances and $\vec{\gamma}$ represents the vector of cell-criticalities. Initially, $\vec{\gamma}$ is populated with switching activity factors (no cells are critical). When static timing analysis, performed between placement iterations, indicates that cell i lies on a critical path (violates a timing constraint), the cell's criticality must be increased $\gamma_i = \gamma_i(1+\delta)$, (along with the weights of critical nets in Φ).

6.6 Empirical Validation

Our implementation of ComPLx inherits the performance and runtime advantages of SimPL (Chapter III). Experiments ran on a 2.8GHz Intel Core-i7 860 Linux server with 8GB RAM, using one CPU core. Detailed placement was done by FastPlace-DP [112]. All settings were the same for all benchmarks. In this chapter, we do not perform empirical comparisons with MAPLE (Chapter V) because MAPLE includes SimPL and its local optimization techniques can be invoked after ComPLx.

6.6.1 Evaluating the core placer

Self-consistency (Formula 6.11). To establish the self-consistency of $P_{\mathcal{C}}$ it suffices to independently establish the self-consistency of each horizontal and vertical pass [86, Section 4] in each room of the floorplan (see below). While the overall algorithm described above using alternating passes differs from LAL in [86], the results produced are essentially the same. The argument for self-consistency remains valid when the algorithm is applied multiple times.

Since the self-consistency condition of Formula 6.11 is transitive, we checked it between every two consecutive ComPLx iterations. Our implementation of the approximate feasibility projection $P_{\mathcal{C}}$ was self-consistent 96.0% and inconsistent 0.6% of the time, while the sufficient condition $||(\vec{x},\vec{y}) - P_{\mathcal{C}}(\vec{x},\vec{y})||_1 > ||(\vec{x}',\vec{y}') - P_{\mathcal{C}}(\vec{x},\vec{y})||_1$ for successive iterations was not satisfied only 3.3% of the time. Thus, the approximate feasibility projection $P_{\mathcal{C}}$ used by our implementation is approximately self-consistent. The convergence plots in Figure 6.1 do not show any disruptions that one would expect with a seriously inconsistent $P_{\mathcal{C}}$. Inconsistencies mostly occur in the early global placement iterations (< 5) where feasible placements differ significantly between consecutive iterations.

Attempts at further improvement. Given that quadratic optimization in SimPL and ComPLx is optimal, we tried to improve the feasibility projection $P_{\mathcal{C}}$. One such attempt used the finest grid during all global placement iterations. In a second attempt, we post-processed the result of $P_{\mathcal{C}}$ by the detailed placer [112] at each iteration. Our data in Table 6.1 show only a marginal improvement, but at a runtime cost. *Vice versa*, coarsening the grid speeds up $P_{\mathcal{C}}$ without undermining solution quality. Thus, no interconnect op-

BENCHMARKS		BEST PUBLISHED as of 02/27/2012	COMPLX					
			FINEST GRID		P_C+=FASTPLACE-DP		DEFAULT CONFIG.	
size (# of modules)		HPWL (placer)	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
ADAPTEC1	211K	**77.82 (RQL)**	79.11	3.86	79.94	93.52	**77.73**	**3.09**
ADAPTEC2	255K	**88.51 (RQL)**	**88.66**	5.23	90.89	162.07	**88.84**	**4.31**
ADAPTEC3	452K	207.67 (SimPL)	206.89	11.54	204.82	323.04	**203.55**	**10.75**
ADAPTEC4	496K	186.80 (SimPL)	**183.87**	10.90	188.37	252.40	**183.16**	**9.57**
BIGBLUE1	278K	**94.98 (RQL)**	96.12	7.22	**94.78**	140.27	**94.41**	**7.00**
BIGBLUE2	558K	**145.47 (SimPL)**	**145.05**	9.91	146.32	200.56	**145.39**	**8.53**
BIGBLUE3	1.10M	**323.09 (RQL)**	352.51	32.52	327.49	629.23	330.74	**24.80**
BIGBLUE4	2.18M	797.66 (RQL)	**788.30**	47.87	792.26	961.25	**788.30**	**41.89**
Geomean		1.00×	1.01×	1.16×	1.00×	26.56×	1.00×	1.00×

Table 6.1: Legal HPWL (×10e6) and total runtime (in min.) comparison on ISPD 2005 benchmarks. Each run uses a single thread on a 2.8GHz workstation. Best-published numbers are annotated with the placers that produced them – SimPL [86] or RQL [141]. mPL6 and NTUPlace3 were included this comparison. We regenerated placements of SimPL without a cell-orientation optimization.

timization during P_C is required. While surprising, this is consistent with the discussion in Section 6.3 and can be explained by the known convergence properties of Primal-dual Lagrange optimization [6, 15]. This decreases the risk of incorrect implementation.

Scalability. Figure 6.3 plots the final values of λ (solid red line) and the number of global placement iterations of ComPLx (dotted blue line with a greater range) on ISPD 2005 and 2006 benchmarks. The number of iterations correlates with the final λ value because each iteration increases λ by a limited amount until the final value is reached. All final values in our experiments are well below 1.0, and the iteration counts do not grow systematically with the size of the input. This phenomenon is consistent with the rapid convergence for which primal-dual Lagrange optimization is known. Given that ComPLx spends near-linear time $O(n(\log n)^p)$ per iteration [86], the overall runtime is near-linear as well. In comparison, the runtime of FastPlace is estimated as $\Theta(n^{1.38})$ [107].

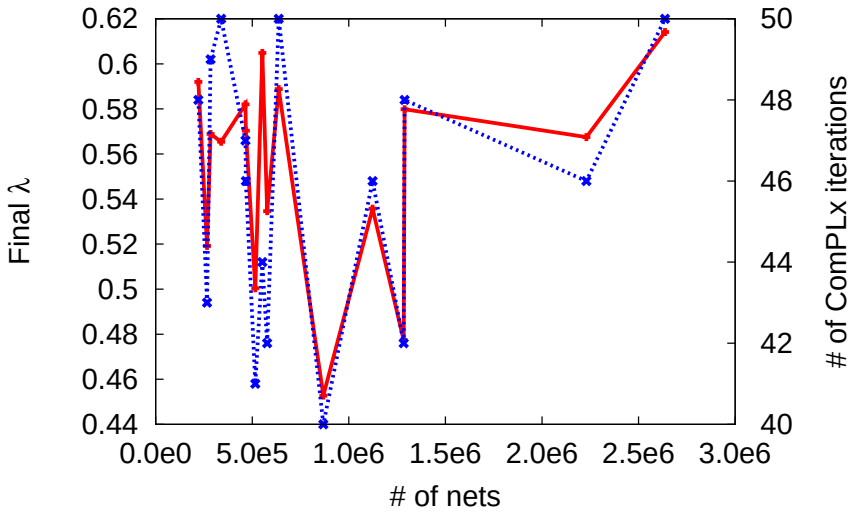


Figure 6.3: The final λ and total number of ComPLx iterations performed, against the number of nets.

6.6.2 Direct comparisons to prior art

On ISPD 2005 benchmarks, ComPLx outperforms SimPL, sometimes by a small amount, sometimes significantly. The similarities are not surprising because ComPLx generalizes SimPL. The improvements are due to the refined convergence criterion (Section 6.3) and improved scheduling of λ (λ corresponds to the pseudonet weight in [86]). ComPLx outperforms SimPL and RQL (the best-published placers excluding MAPLE [87]) by 1%. ComPLx produces best results on more benchmarks than any prior placer, while running 10% faster than FastPlace (including FastPlace-DP runtime in both cases).

Table 6.2 covers ISPD 2006 benchmarks, which include density constraints and movable macros, not handled by SimPL. ComPLx outperforms best-published placer RQL [141] (excluding MAPLE) by 3% in terms of scaled HPWL (the official contest metric). ComPLx is about 12% faster than FastPlace (including FastPlace-DP runtime in both

Benchmarks (Υ_{target})	NTUPL3 [36]	MPL6 [26]	RQL [141]	COMPLX
ADAPTEC5 (0.5)	451.22 (21.0)	431.27 (1.09)	443.28 (9.25)	**415.13 (1.92)**
NEWBLUE1 (0.8)	**62.65 (1.09)**	68.08 (0.14)	64.43 (0.34)	64.75 (1.02)
NEWBLUE2 (0.9)	205.45 (2.53)	201.85 (1.52)	199.60 (1.45)	**194.39 (1.03)**
NEWBLUE3 (0.8)	277.87 (0.00)	284.11 (0.59)	**269.33 (0.07)**	273.42 (0.93)
NEWBLUE4 (0.5)	306.56 (13.1)	300.58 (1.63)	308.75 (15.2)	**292.82 (1.44)**
NEWBLUE5 (0.5)	509.71 (9.56)	537.14 (1.42)	537.49 (13.6)	**507.85 (1.78)**
NEWBLUE6 (0.8)	520.31 (8.40)	522.54 (1.40)	515.69 (4.33)	**501.97 (1.15)**
NEWBLUE7 (0.8)	1109.6 (5.32)	1084.4 (1.14)	1057.8 (2.57)	**1041.4 (1.42)**
Geomean	**1.03**× (2.40)	**1.04**× (1.22)	**1.03**× (2.30)	**1.00**× (1.29)

Table 6.2: Comparison of scaled HPWL (×10e6) on ISPD 2006 benchmarks. Overflow penalties are reported in parentheses. RQL results are from [141].

cases), as well as 6.88× and 8.47× faster than NTUPlace3 and mPL6, respectively. RQL is 3.1× faster than mPL6 [141], hence $> 2.5\times$ as slow as ComPLx (including detailed placement by FastPlace-DP).

6.6.3 Extensibility of the core placer

Practical use of standard-cell placers requires multiobjective optimization and support for design constraints.

Region constraints. Chip designers often impose a region constraint on a subset of cells to express logic hierarchy and clock domains, to keep clock sinks close to clock drivers, or to assist the placer in dealing with challenging critical paths. Traditional placers convert the hard region constraints to soft constraints, which can be addressed by heavily-weighted fake nets [4, Figure 5] or modification of the objective function [40]. While ComPLx sup-

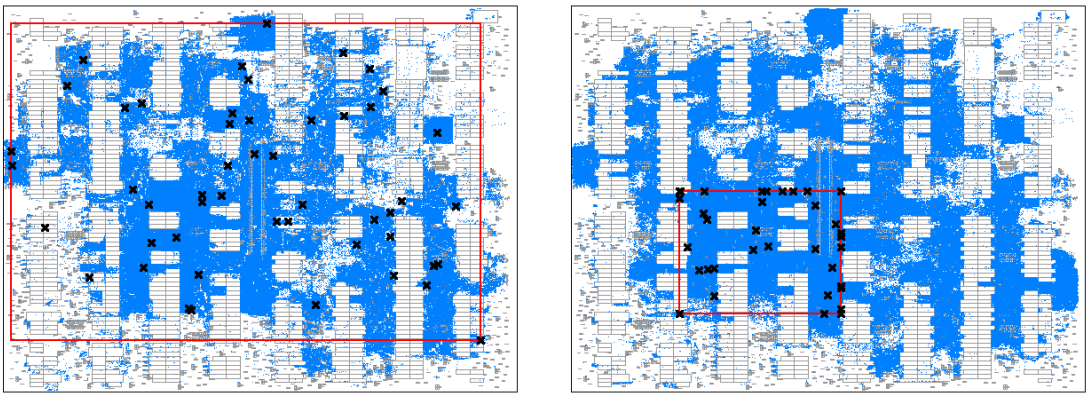


Figure 6.4: A hard region constraint imposed on 50 cells that were initially placed unconstrained (left). The resulting ComPLx placement (right) satisfies the constraint. HPWL drops from 145.39 to 144.57.

ports such techniques, it also allows for a more straightforward and robust implementation of region constraints by enforcing them as part of the feasibility projection at every global placement iteration – each cell is *snapped to* the constraining region after feasibility projection for density constraints. Figure 6.4 illustrates the enforcement of region constraints by "before" and "after" pictures. The locations from the modified feasibility projection are then used as anchors to influence the subsequent iteration of analytic global placement. Rather than degrade, HPWL actually improves — a surprising phenomenon often observed with industry placers.

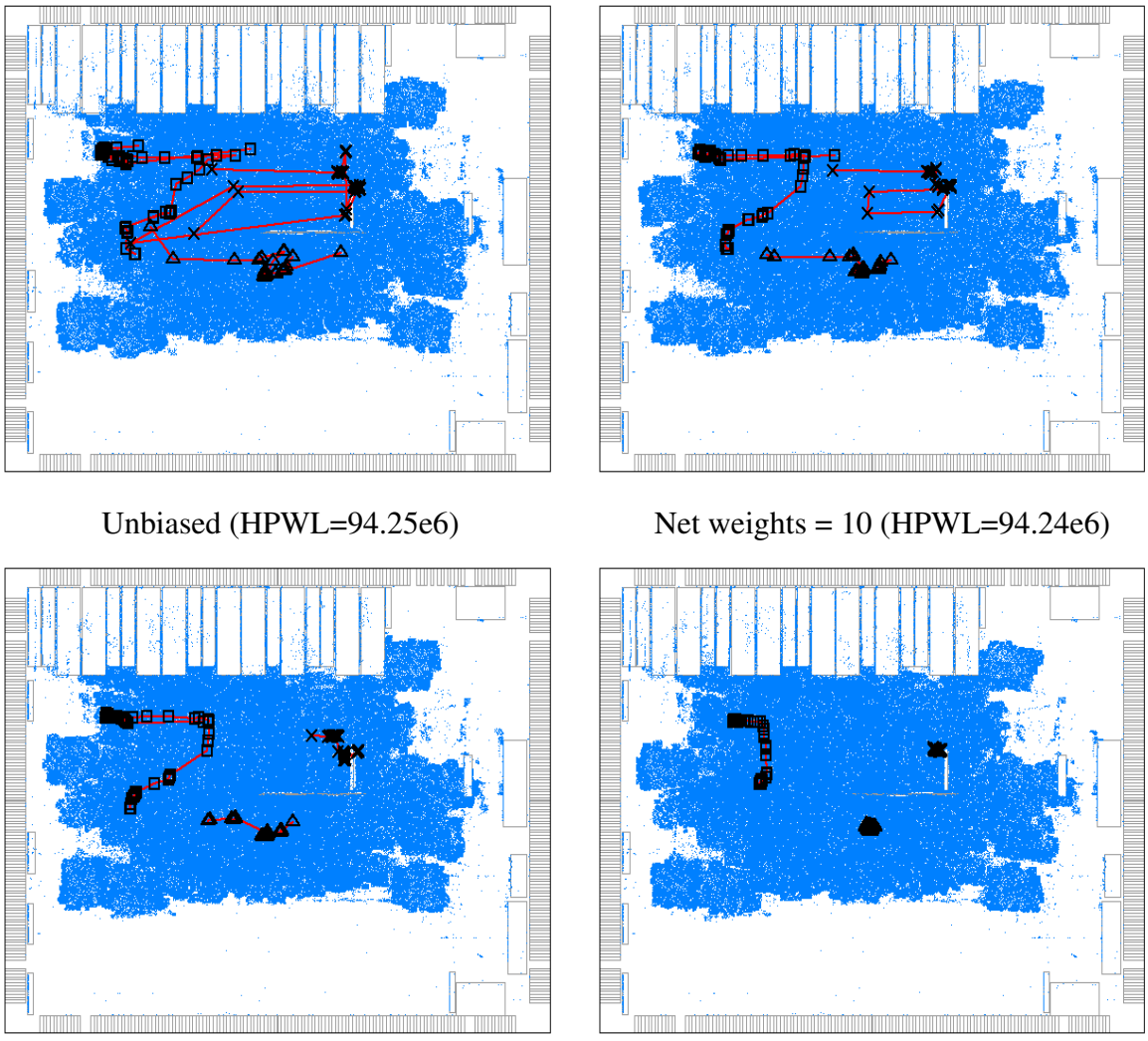
Handling timing-critical nets. To demonstrate effective timing optimization, we show that timing-critical paths can be shortened and straightened by manipulating net weights without adverse effects on total HPWL. Working with the standard benchmark BIGBLUE1, we performed 30 global iterations to obtain an unbiased, stable intermediate placement that allowed us to estimate net lengths. We then selected several critical paths, increased the weights of nets comprising these paths, and ran our placer to completion in three

configurations with different net weights. Figure 6.5 shows that the desired outcome was achieved. With sufficiently large net weights, selected paths notably shrunk. Given that only a small fraction of net weights were modified, the overall placement and its wirelength were largely unaffected. Essential for these results was our scheduling of λ in Formula 6.12. While not a full-fledged demonstration of timing-driven placement, this experiment confirms that our proposed core placement algorithm is capable of controlling critical paths without tangible overhead in HPWL. Specific formulas for provably-good timing-driven net weighting can be found in [24]. Our experiment also demonstrates the *stability* of ComPLx to small netlist changes, which is important in the context of physical design [4].

6.7 Summary and Conclusions

We developed a global placement algorithm ComPLx based on *subgradient projected primal-dual Lagrange optimization*. In its basic form, it consists of (i) interconnect optimization, (ii) a feasibility projection $P_{\mathcal{C}}$ that represents placement constraints, (iii) a penalty term that includes the Lagrange multiplier λ. Our extensions for mixed-size placement handle macros through the feasibility projection $P_{\mathcal{C}}$ and establish a separate, larger λ parameter for each macro. Timing-driven extensions track separate λ for timing-critical cells and increase λ based on criticality (slack). Our baseline algorithm generalizes recent SimPL [86], SimPLR [85] and Ripple [57] algorithms and inherits their empirical success. *Vice versa*, ComPLx provides mathematical substantiation and convergence analysis for SimPL, SimPL and Ripple, suggesting improvements and algorithmic extensions.

A key difference from most prior analytical frameworks is in the spreading mechanism — rather than estimate *density gradients based on local information*, we use a global



Unbiased (HPWL=94.25e6) Net weights = 10 (HPWL=94.24e6)

Net weights = 20 (HPWL=94.15e6) Net weights = 40 (HPWL=94.13e6)

Figure 6.5: In a ComPLx placement of BIGBLUE1 (upper left), three critical signal paths between registers are chosen. Subsequent ComPLx runs are performed with progressively larger net weights on those paths, which straightens the paths and reduces their lengths. Legal HPWL values are reported in parentheses.

feasibility projection P_C. Consequently, the handling of *region*, *alignment* and other types of constraints requires only the modification of the feasibility projection. Avoiding local gradients also improves runtime (compared to APlace and NTUPlace3), and so does our avoidance of optimization by local search (compared to FastPlace and RQL). The tradeoff between spreading and interconnect optimization is controlled by Lagrange multipliers λ.

A key difference from placers based on nonconvex optimization [26,36,79] is the emphasis on decomposing the original problem into convex optimizations, which enables duality and accelerates convergence. Unlike prior works limited to a single interconnect model, our technique can be used with *quadratic*, *log-sum-exp* and other models.

CHAPTER VII

SAPT: Structure-Aware Placement Techniques[1]

This chapter demonstrates that conventional Half-Perimeter Wirelength (HPWL)-driven placers underperform in terms of Steiner Wirelength (StWL) for hybrid designs, wherein designs contain a mixture of random logic and datapath standard cell components. The quality gap between manual placement and automatic placers is more pronounced in the presence of datapaths. To effectively handle hybrid designs, we develop a new unified placement flow that simultaneously places random-logic and datapath standard cells. This flow was developed during an internship at IBM Research and relies on the SimPL placer introduced in Chapter III. It significantly improved datapath layouts and placement quality of mixed netlists designed at IBM while leveraging the speed and flexibility of the existing random-logic placement algorithm. It consists of a set of novel global placement techniques, collectively called Structure-Aware Placement Techniques (SAPT). These techniques effectively integrate alignment constraints into placement, thereby overcoming the deficiencies of existing random-logic placers when handling designs with embedded datapaths. Experimental results comparing our placement flow with six state-of-the-art placers

[1]This work is an extension of [151] and was developed jointly with Samuel I. Ward, who made an equal contribution to it.

on the ISPD 2011 Datapath Benchmark Suite released by IBM Research show at least a 28% improvement in total StWL with over a 6 times improvement in total routing overflow. On industrical hybrid designs that integrate datapath into larger netlists, we report a 5.8% improvement in total StWL

7.1 Introduction

As SoC frequencies exceed 1 GHz and shrinking schedules drive increased automation for microprocessor designs, the boundary between manually designed datapath logic and random-logic macros is blurring. A new *hybrid* design style is emerging, wherein designs contain both random logic and datapath. The datapath logic generally refers to circuit structures containing highly parallel bit operations [109] (often called the bit-stack), and careful design is important for high frequency designs. Prior work [68] has shown that handling the datapath logic placement independent of the random logic, overly constrains the random-logic placement, degrading overall congestion and wirelength. A single placement flow handling both structures is extremely valuable, improving design time, solution quality, and saving development and maintenance costs. However, [113, 149] demonstrate that most state-of-the-art placers are incapable of handling designs with regular structure.

This chapter shows that with a small amount of design guidance, existing HPWL-driven placers can be guided to better handle designs with embedded datapath logic. The clue is disregarding placement of high fanout nets that can disrupt the structure of the datapath. Most placers will create a clique with a low weight to model these nets, or treat them just like other low fanout nets. To minimize HPWL of a high fanout net, a placer would naturally compact the placement of them in all directions – which is the exact opposite of

what is required by placement of regular datapath logic structures, as shown in [91]. To this end, we propose a novel structure-aware placement flow for hybrid designs via a set of effective placement techniques that can be incorporated within existing random-logic placers. The proposed flow leverages the speed and flexibility of state-of-the-art HPWL-driven placers, while imposing alignment constraints to achieve better regularity and StWL.

The key contributions of this chapter are as follows:

1. A study of the issues with current academic placers: the inadequacies and specifically the lack of fidelity of the HPWL model versus the StWL model when evaluating and placing datapath logic.

2. A key insight to bit-stack alignment: alignment of the bit-stack guides indirect StWL optimization, and significantly improves total StWL and routing congestion.

3. A novel placement flow: Structure-Aware Placement Techniques (SAPT) that can be incorporated within existing HPWL-driven placers to enable better alignment of the embedded datapaths during both global and detailed placement.

Section 7.2 outlines the problem faced by current random-logic placers when placing datapath logic. Section 7.3 provides an overview of the proposed flow with general descriptions of each technique. Our structure-aware global placement techniques are described in Section 7.4 and structure-aware detailed placement techniques are described in Section 7.5. Experimental results are presented in Section 7.6. A summary and conclusions are given in Section 7.7.

7.2 Motivation and Background

A common assumption among IC designers is that circuits with high regularity such as datapath logic require manual placement. Perpetuating this assumption are two key factors that limited adoption of past automation attempts. First, prior approaches separate control logic placement from datapath logic placement. Second, a prevailing evaluation metric for random-logic placement, HPWL is inadequate for structured circuit styles. This section addresses each of these factors by first demonstrating the need for a unified placement framework and then by diagnosing the inadequacy of the HPWL metric for regular structures. Then, it validates that cell alignment during placement implicitly optimizes StWL producing significant wirelength improvements for datapath style circuits.

7.2.1 The need for a unified placement framework

Automatic placement of structured circuits has been performed by dedicated datapath placers such as [109, 131, 164], which generate highly compact, area efficient placements. Once the datapath layout is generated, these methods construct a larger macro block or small individual bit-slice macro blocks. These macro blocks are then placed by the main random-logic mixed-size placer. The primary drawback of these approaches is that even though a datapath placer may minimize the local wirelength through cell ordering [54] or optimizing specific bit-stacks [161], global interconnect optimization with the embedded datapath is not taken into account during placement. As shown by [68, 113, 149], the constraints added in this process produce suboptimal results in practice.

	ISPD Datapath Benchmark A					ISPD Datapath Benchmark B			
	Total HPWL		Total StWL			Total HPWL		Total StWL	
Manual	11000365	1.00	11066683	1.00	Manual	8642097	1.00	9823680	1.00
Capo	11535525	**1.05**	21516128	1.94	Capo	10338805	**1.20**	23881606	2.43
SimPL	11837307	1.08	20180311	**1.82**	NTUPl3	10433894	1.21	26110039	2.66
mPL6	12919955	1.17	23950663	2.16	SimPL	10631304	1.23	22319594	**2.27**
NTUPl3	13447753	1.22	24673151	2.23	Dragon	12229019	1.42	28577316	2.91
FastPl3	15672727	1.42	27115750	2.45	FastPl3	14537026	1.68	36642434	3.73
Dragon	16424739	1.49	26182449	2.37	mPL6	16263018	1.88	28846387	2.94

Table 7.1: Legalized HPWL and StWL comparison on the ISPD 2011 Datapath Benchmark Suite [150] between manually placed and automated placement solutions. Placement results are sorted by increasing HPWL value. The best HPWL solution does not indicate the best StWL solution. NTUPl3 stands for NTUPlace3 and FastPl3 stands for FastPlace3.

7.2.2 StWL and HPWL comparisons for datapath circuits

The second key factor limiting datapath placement automation is that traditional HPWL-driven placers naturally compact the placement of high-fanout nets to reduce total HPWL of circuits. However, known optimal layouts for many regular datapath structures are drastically different [91]. This implies that the mere HPWL objective for placement on highly regular circuits is inadequate because the HPWL metric can significantly underestimate routed wirelength. To illustrate this point, Table 7.1 compares the placers mPL6 [26], Capo v10.2 [121], FastPlace v3.0 [140], NTUPlace3 v7.10.19 [36], Dragon v3.01 [137] and SimPL [86] using both, total Half-Perimeter Wirelength (HPWL) and total Steiner Wirelength (StWL) on the modified ISPD 2011 Datapath Benchmark Suite [149, 150].[2] For improved experimental control, all StWL measurements were performed using coalesCgrip [41], and all reported numbers are total wirelength results for each design. The HPWL column in Table 7.1 is sorted from smallest to largest for each benchmark. The

[2]The MISPD 2011 Datapath Benchmark Suite was modified to contain unfixed latch rows compared to the original fixed latch placement reported in ISPD 2011. Benchmarks can be downloaded at: http://www.cerc.utexas.edu/utda/download/DP/

table reports the measured HPWL and StWL values for the benchmark circuits as well as the wirelength ratio normalized to the manually placed solution. Careful examination of this table yields the following surprising results:

1. While HPWL from the automated placement solutions for both benchmarks is very close to the manually placed solution, the StWL results degrade significantly, with the best automated solution at 1.82 times in StWL for benchmark A and 2.27 times for benchmark B compared to the manual solution.

2. Fidelity of the HPWL metric appears low for datapath logic. As shown in Table 7.1, the HPWL column is sorted by increasing value and it is generally expected that StWL would maintain the same order. But in fact that does not happen. However, in both cases, the placer with the best HPWL does not generate the best StWL. For benchmark A, though Capo generates the best HPWL (5% larger than the manually placed solution), SimPL generates the best StWL (1.82 times higher than the manual solution). The same holds true for benchmark B. Again, Capo generates the best HPWL, but SimPL generates the best StWL (2.27 times higher than the manual solution).

As shown in Section 7.6.4, the significant improvement in StWL also corresponds to vastly improved congestion metrics. There has been prior work in directly optimizing StWL [123]. As reported in [123], StWL has much better correlation to the routed wirelength (rWL) as compared to HPWL. However, the wide-spread adoption of the HPWL metric empirically shows the effectiveness of the model due to its easy modeling and good first-order approximation to timing and power on vast number of random-logic designs.

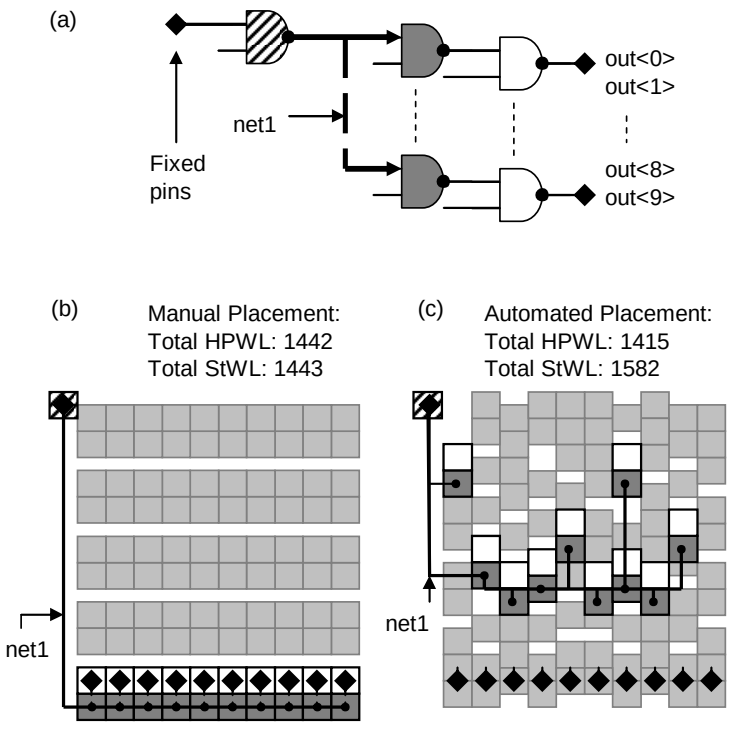


Figure 7.1: An example circuit where StWL of the manually placed design is better than that of the automated placement, but HPWL of the automated placement solution is better than that of the manual placement. Net1 has fanout of 10.

Therefore, instead of completely changing the placement objectives, we seek methodologies to improve placement quality of datapath logic under the hood of existing HPWL-driven placement frameworks.

7.2.3 Implicit StWL optimization through bit-stack alignment

Figure 7.1 provides an implication on how to accomplish this. In Figure 7.1a, a partial logic netlist with one NAND gate, shown as hashed, drives net $net1$ with a fanout of 10. All the input and output pins are fixed objects placed on top of the gate. Figure 7.1b shows a manually placed solution for this partial circuit and Figure 7.1c shows a solution from an existing placer. The dark-shaded cells match the same dark-shaded NAND gates in Figure 7.1a. The light-shaded gray cells represent other logic placed within the design.

For both solutions, we measure the total HPWL and StWL, and the numbers are shown in Figure 7.1. As pointed out in Section 7.2.2, even though HPWL of the manual solution (1442) is greater than HPWL of the automated placement (1415), StWL shows the reverse trend. While it is impractical to list HPWL and StWL of every single net, clearly for net $net1$, the StWL number in Figure 7.1b is better than the StWL number in Figure 7.1c. This is due to the better alignment of the structured cells in one horizontal row, which produces much better StWL. Also the solution of Figure 7.1c shows the existing placer compact the placement of the net in both x- and y-directions to lower HPWL, but degrading StWL. As this example shows, if a HPWL-driven placer can obtain better alignment for regular structures, it will have better StWL implicitly without having to directly optimize StWL.

Motivated by the above examples, we develop new techniques to guide an existing random-logic placer to generate a placement solution similar to Figure 7.1b, with better StWL than the one in Figure 7.1c. Additionally, by providing alignment constraints to small portions of the datapath, we observe that during the iterative placement process, other surrounding cells become aligned as well. This can be observed visually in the placement results in Figure 7.8 where only some of the cells have been manually defined. Previously explored approaches, like post ECO datapath placement, or placing the datapath as a macro block, tend to ignore the connections between random logic and datapath cells since they place every datapath cell *a priori*. The alignment constraints presented in this chapter however are providing hints to placers, directing them toward more globally-optimized solutions. As results will show, with relatively few manually defined bit-stacks, our framework significantly reduces overall wirelength and congestion.

7.2.4 Alignment groups

We first define *alignment group* g_k, as follows.

Definition An alignment group $g_k \in G$ where $0 \leq k < |G|$, is an unordered subset of cells from V. An alignment direction $\vec{d_k}$ is a preferred placement direction of g_k where $0 \leq \vec{d_k} \leq 90$. The collection of g_k with the same d_k value is pairwise disjoint.

Generally, g_k may correspond to bit-stacks in the datapath, but can be other elements such as cells connected to a single high fanout net that improves through alignment, buffers that need careful placement to facilitate routing of large buses, or pipelining latches. For alignment directions, we only horizontal and vertical directions, i.e., $\vec{d_k} \in (0, 90)$. In the example shown in Figure 7.1, $\vec{d_k} = 0$ for NET1. In this work, we assume that a set of alignment groups G and their alignment directions are given. This assumption is valid and practical; One may use datapath extractors such as [38,118] to extract the alignment groups based on circuit properties. Alternatively, this information may come directly from logical descriptions of netlist, or could be provided by designers. As an example, if designers are trying to structure the latch placement to be aligned, it is trivial for them to provide sets of latch names and their preferred placement directions (horizontal or vertical).

7.2.5 Alignment net insertion

To generate better datapath alignment, one approach is direct manipulation of existing nets between the datapath cells. However, this approach interferes with other prior placement enhancements. Specifically, direct weighting manipulation of current nets disrupts timing- or power-driven placement and net weighting for those cells. Due to the

above problem, a new method is instead proposed, and a new category of nets, referred as *alignment nets* is defined.

Definition An alignment net s_k where $0 \leq k < |G|$, is a weighted multi-pin connection between all cells in an alignment group g_k. For placement, the net weight for this alignment net is modeled by the Bound2Bound net model [134].

Alignment nets are created at the beginning of global placement and remain persistent during the entire global and detailed placement stages. A skewed net weight scheduling (Section 7.4.1) helps these nets align the cells within the corresponding alignment group g_k inside the layout region. By applying alignment constraints to new nets s_k, prior techniques relying on net weighting continue to function as before.

7.3 Unified Placement Flow with Alignment Constraints

We employed the SimPL framework (Chapter III) as a baseline. The proposed placement flow is presented in Figure 7.2, where the white boxes highlight the enhancements applied to each g_k and the shaded boxes are the original flow. Though we used the SimPL framework, the techniques can be adapted to other force-directed placement frameworks as well. We refer to these structure-aware placement techniques as SAPT. The key algorithmic components of SAPT from the flow diagram are:

1. *Alignment Net Insertion:* We insert alignment nets to manipulate the placement solution of specific cells during global placement. (Section 7.2.5)

2. *Alignment Net Order Extraction:* On highly structured circuits, it is possible to extract specific optimal order between alignment groups. We present a process to

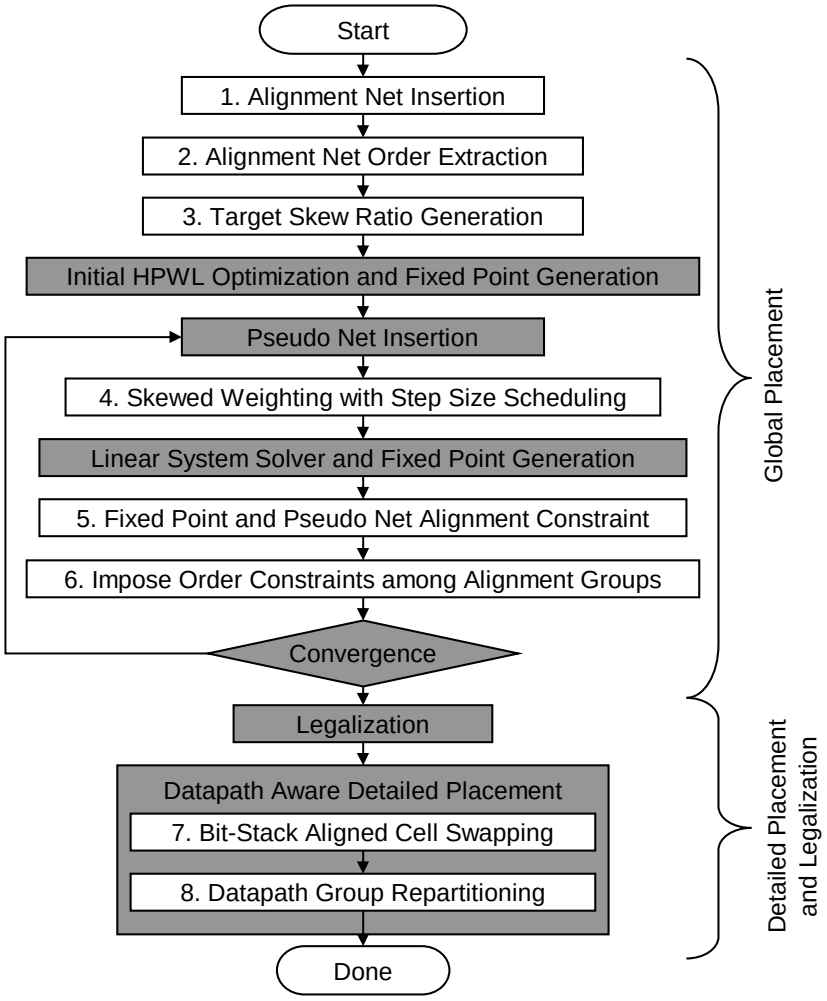


Figure 7.2: Proposed datapath-aware placement flow. The baseline components are shown in transparent boxes and the added datapath-aware components are shaded.

automatically extract the relative order between alignment groups if they exist prior to placement. (Section 7.4.5)

3. *Target Skew-Ratio Generation:* This section describes the process for generating the specific ratio for the skewed weighting process as described in Section 7.4.1. It describes two key insights to effectively select the skew ratio and proposes a model for automatic generation. (Section 7.4.3)

4. *Skewed Weighting with Step Size Scheduling:* This describes the skewed weighting process applied to each alignment net s_k to gradually improve alignment along the datapath. The weight in a particular direction gradually increases during each placement iteration until it reaches the target skew (described in Section 7.4.3) for that alignment net (Section 7.4.1).

5. *Fixed-Point and Pseudo Net Alignment Constraint:* Modern force-directed global placement frameworks reduce overlap between cells by using fixed-points and pseudo nets. For datapath logic, this process leads to misalignment. This section describes a process to modify the fixed-point location minimizing wrong-way perturbations (Section 7.4.4).

6. *Impose Order Constraints between Alignment Groups:* Fixed-point alignment constraints generate a relative order between alignment groups that is not always optimal. Therefore, this section proposes a method for improving the order between alignment groups during global placement and show how to improve their global position. (Section 7.4.5)

7. *Detailed-Placement Techniques:* We employ (i) *Bit-Stack Aligned Cell Swapping* and *Alignment-Group Repartitioning* from [151] for the structure-aware detailed placement (Section 7.5).

At each step, the modifications apply only to the defined alignment groups g_k leaving all other random-logic cells to be placed as they would before. We provide detailed descriptions of the global and detailed placement techniques in Sections 7.4 and 7.5.

7.4 Structure-Aware Global Placement

This section presents our global placement techniques for providing alignment and ordering constraints to cells. We first present skewed weighting with step size scheduling and then we propose a method for generating the target skew automatically. Then we outline methods to force alignment by modifying fixed-point locations and complete this section by showing how to extract the order for those fixed-point locations.

7.4.1 Skewed weighting with step size scheduling

In this section, we introduce a skewed weighting technique that encourages alignment of s_k in preferred placement directions $\vec{d_k}$. The high level idea is to apply gradually increasing net weights for s_k only in orthogonal directions to $\vec{d_k}$. This application of higher net weights in one particular direction (i.e., skewed weighting) increases corresponding costs (HPWL($\vec{x}$) or HPWL ($\vec{y}$)) of a quadratic objective. Consequently, the linear system solver generates compact placement in unpreferred directions to reduce the overall cost. The rate of change of the weighting value increases slowly during the initial stages of global placement, increases rapidly during the middle stages, and slows again near the end of global placement. This is preferable to applying hard constraints (forced alignment) in the early stage of wirelength optimization, as it can disrupt the original optimization and often lead to a solution that suffers from sub-optimality in terms of overall wirelength.

7.4.2 Step size scheduling

Let n be the current global placement iteration and M be its upper bound,[3] and we define $p(n)$ as the alignment weight schedule function for each iteration n. The following

[3] M is typically set to 50 [86]

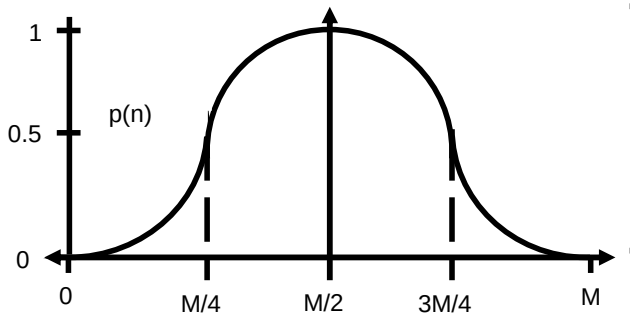


Figure 7.3: Bell-shaped step size scheduling function.

equation for $p(n)$ is proposed:

$$p(n) = \begin{cases} \frac{8n^2}{M^2} & 0 \leq n < \frac{M}{4} \\ 1 - \frac{8(n-\frac{M}{2})^2}{M^2} & \frac{M}{4} \leq n \leq \frac{3M}{4} \\ \frac{8(n-M)^2}{M^2} & \frac{3M}{4} < n \leq M \end{cases} \tag{7.1}$$

To avoid imposing hard constraints during the early iterations of global placement, $p(n)$ gradually increases during the initial iterations and to minimize large constraint changes during the final stages, the function decreases towards zero at the last iteration. This function is also used in [79] to model cell density, but it serves a *completely different purpose* here as a scheduling function. Using $p(n)$, Equation 7.2 displays the skewed monotonically increasing weighting parameters γ^n and δ^n for alignment net s_k. Using $p(n)$ directly generates very large weighting steps therefore a constant scaling factor β is added. This parameter is left default throughout all placement runs. Let $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ be the directional unit vectors and $\sigma^2_{x,y}$ the nth iteration's variance in either the x or y direction. Finally, the modified placement equation is shown in Equation (7.3). For non-alignment nets, $\delta_{i,j} = 0$ and $\gamma_{i,j} = 0$.

$$\begin{aligned} \gamma^n &= \gamma^{n-1} + \hat{\mathbf{y}} \cdot \overrightarrow{\mathbf{d_k}} * \beta * p(n) * \sigma^2_x(n) \quad \text{where } \gamma^0 = 1 \\ \delta^n &= \delta^{n-1} + \hat{\mathbf{x}} \cdot \overrightarrow{\mathbf{d_k}} * \beta * p(n) * \sigma^2_y(n) \quad \text{where } \delta^0 = 1 \end{aligned} \tag{7.2}$$

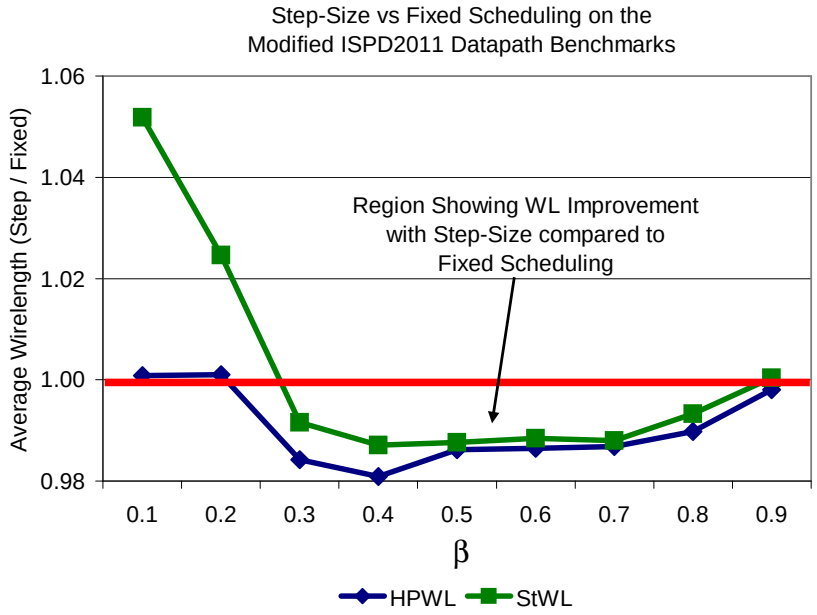


Figure 7.4: Average wirelength improvement using the proposed bell-shaped step-size scheduling function on the ISPD2011 Datapath Benchmark Suite at different values of β.

$$\Phi_G(\vec{x},\vec{y})^n = \sum_{i,j}[(\gamma^n_{i,j} + w_{i,j})(x_i - x_j)^2 + (\delta^n_{i,j} + w_{i,j})(y_i - y_j)^2] \tag{7.3}$$

Figure 7.4 displays the average change in wirelength between fixed scheduling and the bell-shaped step-size scheduling using different values of β. It shows that for $0.2 < \beta < 0.9$, the bell-shaped step-size scheduling yields on average, better HPWL as well as StWL. In our implementation we used $\beta = 0.5$ based on empirical data.

7.4.3 Target skew-ratio generation

In section 7.4.1, a method is proposed to systematically increase the skewed weight for an alignment net by either increasing $\gamma^{i,j}$ or $\delta^{i,j}$ as placement evolves. However, in our prior work [149], the final values for γ^n or δ^n were provided as inputs, primarily based on the designers' discretion. This section extends this shortcoming by presenting key insights for automatic selection of target skew ratios and proposes a method for automatically

generating the target skew. We first define *target skew ratio* and *target skew multiplier*.

Definition Target skew ratio is the final ratio γ^n/δ^n of the net weighting in the x-direction and y-direction for an alignment net at the completion of all global placement iterations.

Definition Target skew multiplier sk_i is the amount the original γ^0 or δ^0 alignment net i will have increased to at the completion of global placement to equal γ^n or δ^n. Thus, $\gamma^n = \gamma^0 * sk_i$ and $\delta^n = \delta^0 * sk_i$.

As an example, given a vertically aligned alignment net, during the initial global placement iteration $\gamma^0 = 1$ and $\delta^0 = 1$. However, by the end of global placement iteration n, the X-direction weight would have increased to $\delta^n = \delta^0 * sk_i$ causing the cells within the datapath group to become aligned vertically. We now outline key insights for effective target skew multiplier selection and illustrate them using Figure 7.5. In this figure, the red alignment net is shown connecting each of the datapath cells within a datapath group and the dashed lines indicate net connections between each cell within the datapath group and other cells outside the datapath group.

The first key insight is that the alignment net weight must overcome the interconnect forces imposed by the nets connected between cells within the alignment group. In other words, the force pulling the alignment group into alignment must be greater than the force pulling the cells out of alignment.

To model this force, we propose a modification to rentś rule that relates the internal pin connections on the alignment net versus the external pin connections on the datapath cells. In the past, Rentś rule 7.4 was used to predict interconnect where g is the total number

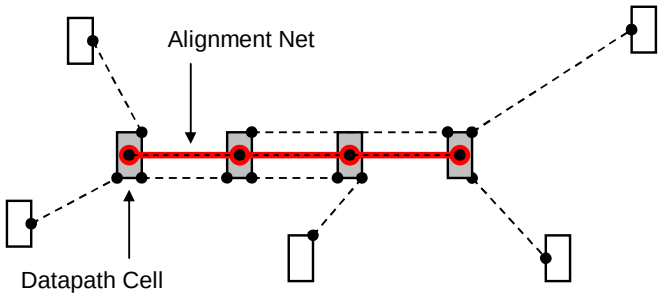


Figure 7.5: Skewed weight force example.

of cells, T is the total number of external connections, t is a design constant and p is the exponent where p is generally $.5 < p < 1$.

$$T = t * g^p \tag{7.4}$$

This chapter proposes the following modifications to the variable definitions and then solve for the p_i value for a particular alignment net i. The variable T equals the total external pins for an alignment group which is the sum of all pins for each cell. This includes pins with a net connecting cells within an alignment group and pins connecting to cells outside of the alignment group. The variable g equals the total number of internal connections which are pins connected to the alignment net. As before, the variable t is a placer specific constant. The resulting p value relates the internal connections on the alignment net to the external connections among the cells. As with the traditional interpretation of p, a larger p value implies more external connections necessitating a higher skewed weight multiplier. Conversely, a smaller p value indicates fewer external connections thus requiring a smaller multiplier.

The second key insight for automatic target skewed ratio selection is consideration of additional interconnect forces from alignment nets themselves, The alignment nets are

employed to encourage very compact placement in unpreferred directions, however, they pull the connected cells together even in $\vec{d_k}$. The HPWL model naturally clumps the cells in alignment nets together, and therefore the target skew ratio must be properly adjusted to offset this side effect. Our key observation is that the required skew ratio should be linearly increased as the cardinality of an alignment group g_i increases. Thus, we define the second component of the target skew multiplier as follows: $\alpha * |g_i|, \alpha > 1$ where α is a placer specific constant that insures for a two-pin net, the two cells are always correctly aligned. For example, if aligning two cells horizontally, the Y-weight must be at least α times larger than the X-weight to guarantee the cells align horizontally.[4]. We set α to a value of 1.25. This constant is then multiplied by the total number of cells in the datapath group. Finally, taking both components into account, the skewed weight multiplier is calculated as shown in Equation 7.5.

$$\xi_i = (1 + log_g(T/t)) * \alpha * |g_i| \tag{7.5}$$

7.4.4 Fixed-point alignment constraint

Modern force-directed global placement frameworks use fixed-points and pseudo nets to discourage cell overlap. By gradually perturbing the unconstrained linear system solver, global placement iterations progressively generate placements with less overlap. In SimPL (Chapter III), at each global placement iteration, lookahead legalization (LAL) generates fixed points that are connected to their corresponding movable cells with two-pin pseudo nets. In the subsequent global placement iteration, these pseudo nets exert pulling forces

[4]This parameter is set when placing only two cells.

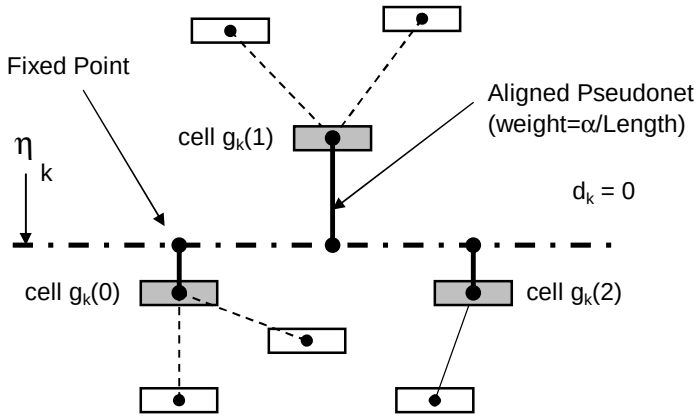


Figure 7.6: Example of a fixed-point alignment constraint for a horizontal bit-stack. Lookahead legalization generates new zero-area fixed-points and the locations of these points are modified to be in alignment with η_k^n.

and reduce the amount of cell overlap. For datapath logic, LAL and subsequent pseudo net insertion step cause misalignment within the bit-stack requiring a constraint forcing alignment which minimizes wrong-way perturbations in the bit-stack.

The proposed fixed-point alignment constraint is applied in two steps. First, LAL generates fixed-point locations for all cells Second, for all cells in datapath group g_k, modified fixed-points are added. We denote a location of this modified fixed-point for cell i as $\eta_{k,i}^n$ for the nth iteration. We compute $\eta_{k,i}^n$ as follows:

$$\begin{aligned} \eta_{k,i}^n &= (x_i, \sqrt[|g_k|]{\prod_{j=1,\ldots,|g_k|} y_j}), \quad \text{if } d_k = 0 \\ \eta_{k,i}^n &= (\sqrt[|g_k|]{\prod_{j=1,\ldots,|g_k|} x_j}, y_i), \quad \text{if } d_k = 90 \end{aligned} \tag{7.6}$$

An example of the modified fixed-point locations and corresponding pseudo nets for a horizontal datapath is shown in Figure 7.6. In this example, there are three gray cells, $g_k(0:2)$ are in one datapath group g_k. The other cell connections are shown with the dashed line connected to the hollow cells. For random logic cells, the fixed-points will be determined by the LAL step alone. For the datapath cells shown, after LAL generates

a new fixed-point locations, those locations are modified based on the geometric mean parallel to the datapath direction $\vec{d}_k$.

Modifying the fixed-point locations enables the global placer to progressively reduce cell overlap while maintaining bit-stack alignment. Two items should be noted about this process. First, we *only modifies the fixed-point location* for datapath logic, not the weighting of the pseudo net. The pseudo net weighting, in contrast to the alignment net weighting proposed in Section 7.4.1, acts on datapath and random logic the same. Second, though this technique violates the overlap constraint during global placement, the cell overlap progressively reduces with global placement iterations [86, Figure 6] and small remaining cell overlap can be easily removed during legalization without undermining the overall solution quality.

7.4.5 Imposing order constraints between alignment groups

In this section, we propose a method for handling the order between alignment groups during global placement and show how to improve their global positions. Even though imposing alignment constraints can line up alignment groups, their relative and absolute positions can be considerably different from custom solutions, which can contribute to the suboptimality of automated placement. Therefore, we first extract the relative order between alignment groups if it exists, and then place them while preserving this order.

Extraction of the order between alignment groups is performed in two stages: (i) extraction of logical orderings between alignment groups based on Depth-First Search (DFS) and (ii) extraction of physical order between logically-independent *supergroups* based on primary input (PI) / primary output (PO) pin locations.

In the first stage of extraction, we identify sets of logically-related alignment groups per $\vec{d_k}$ and determine a unique order within the sets. The process begins with constructing a directed graph based on the netlist and traversing through each defined alignment group. Starting from a cell in an alignment group, DFS identifies a set of logically-related alignment groups, which we denote by a *supergroup*. This way, we can extract multiple logically-independent supergroups where a unique order within each supergroup is defined. Since the resulting order in a supergroup is logical, alignment groups can be either placed from left to right or from right to left. Without loss of generality, we place them left to right.

Unlike other random logic-oriented designs, in datapath designs, there can exist multiple supergroups that are logically independent while having the symmetry in their structure. For example, as seen from SPBA in [149, Figure 4], first m bit MUX select signals form a supergroup, and such supergroups are repeated many times at different pipeline stages. Even though such supergroups are logically independent, and the relevant logic cells around each subgroup drive different PO pins, one can find that they share the same structure. In addition, as discussed in [149], having known that some fixed PI / PO pins can be pre-placed on top of their respective connections to reduce PI / PO routing interconnect, we can further determine the order of supergroups despite their logical independence.

In the second stage of extraction, therefore, we seek the closest PI / PO pins that are connected to each supergroup. Given that such PI / PO locations are ideal to interface with other external logic, their physical locations provide clues for relative positioning of supergroups. By taking advantage of the symmetry of logically independent supergroups,

we find fixed pins that are connected to each supergroup with the same smallest logical distance and determine a unique order of supergroups by referring to these locations.

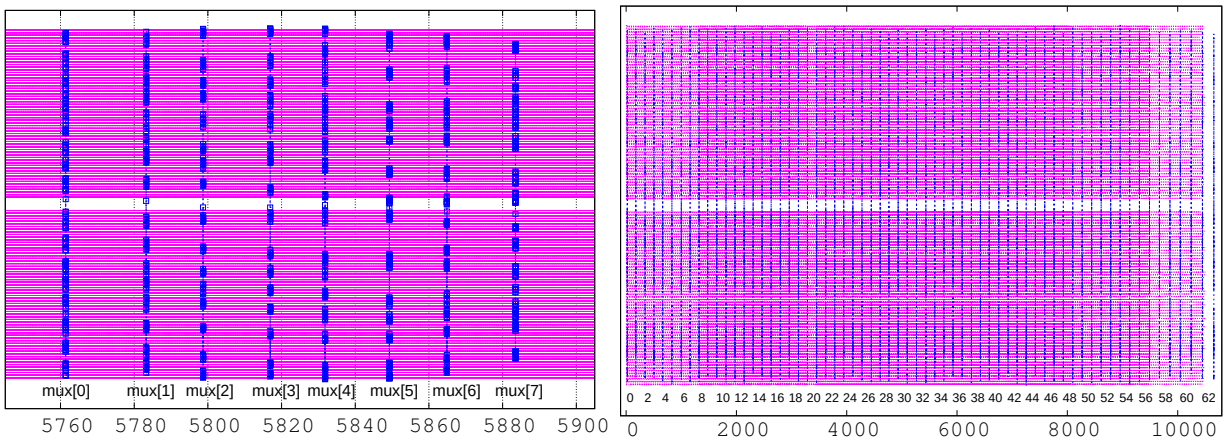


Figure 7.7: Placement of MUX select signals within a supergroup (left) and MUX[0] signals across supergroups (right) on SPBA. Red lines indicate alignment nets with $\vec{d_k} = 0$ while blue dotted lines indicate alignment nets with $\vec{d_k} = 90$. Blue boxes are MUX cells. For both cases, one quarter of the horizontal nets (i.e., alignment nets with $\vec{d_k} = 0$) are shown.

Imposing ordering constraints across alignment groups is achieved by post-processing the fixed-pin locations given by pseudo net alignment constraints (Section 7.4.4). Instead of directly using geometric means to align fixed-point locations, we (i) collect and sort geometric means of all alignment nets by their values and (ii) assign the sorted geometric mean values to alignment nets in order of the extracted ordering of alignment nets. Then, we use the assigned geometric means to align fixed-point locations per alignment net. During early iterations of global placement, this approach may corrupt the spacing between alignment nets if they are not initially uniform. However, once the placement and order begin to stabilize, placement evolves locally and recovers good spaces between alignment groups, as can be seen from Figure 7.7.

Table 7.3 summarizes supergroups and their cardinalities that are extracted via the pro-

posed techniques on the ISPD 2011 datapath placement benchmarks. Alignment groups are defined based on net degrees, which represent select / data input signals. Table 7.2 empirically demonstrates the impact of the proposed techniques on solution quality. The second column only considers the logical order within supergroups and the third column additionally considers the physical order between supergroups. As can be seen from Figure 7.7, the order of alignment groups is preserved during placement.

CKT.	$\vec{d_k} = 0$			$\vec{d_k} = 90$		
	Deg.	# SGs	$\|SG\|$	Deg.	# SGs	$\|SG\|$
SPBA	68	128	1	257	64	8
SPBB	256	257	1	261	512	1

Table 7.2: The net degrees (Deg.) used to define alignment groups, the number of supergroups ($\#SGs$) and their cardinalities ($\|SG\|$) on the ISPD 2011 datapath placement benchmarks. On SPBB, $\|SG\|$ were 1 for both $\vec{d_k} = 0$ and $\vec{d_k} = 90$, which indicates no logical ordering were found between any alignment groups.

CKT.	No ordering constraints		Intra-SG ordering		+Inter-SG ordering	
	HPWL	StWL	HPWL	StWL	HPWL	StWL
SPBA	12.05	15.79	11.74	15.22	11.46	15.06
SPBB	10.58	15.78	10.58	15.78	10.47	15.71
Avg.	1.03×	1.03×	1.02×	1.01×	1.00×	1.00×

Table 7.3: Legal HPWL and StWL (×10e6) comparison varying the ordering constraints. We employed the fixed weighting scheme for the experiments. On SPBB, results in the first two columns are identical since no logical ordering were found.

7.5 Structure-Aware Detailed Placement

In the prior section, we presented techniques for aligning cells during global placement. However, current detailed-placement techniques will disrupt the structure generated during global placement. To maintain aligned placement, we employ two detailed-placement techniques originally proposed in our previous work [151]: (i) *Bit-stack aligned*

cell swapping and (ii) *Alignment-group repartitioning*.

Bit-Stack Aligned Cell Swapping looks for cells to swap between the current location of cell j and all cells within the aligned swap region. If a swap produces improved HPWL, the cell locations are updated. Doing this maintains the alignment generated during global placement. **Alignment-Group Repartitioning** minimizes internal net cut values potentially improving both HPWL and StWL metrics for all nodes in g_k along s_k. The impact of these techniques is reported in Tables 7.5, 7.6, 7.8 and 7.9 in the last rows.

7.6 Experimental Results

This section empirically demonstrates the effectiveness of the proposed techniques by presenting results on two design styles, industrial *hybrid* designs and the highly structured Modified ISPD 2011 Datapath Benchmark Suite [150]. All experiments were run on a 3.2 GHZ Intel(R) Xeon(R), X5672 machine with 96 GB of memory.

The placer is built on the SimPL [86] global placement and FastPlace-DP [112] detailed placement frameworks. It is compatible with the Bookshelf format and requires an additional datapath definition file as input. This file, loaded prior to global placement, includes each cell of the alignment group and the group‘s direction. Since this work focuses on the placement solution, each datapath was manually defined for improved experimental control. To independently quantify the effectiveness of our global and detailed placement techniques, we present results from two flows:

- SAPTgp + FP-DP: Our structure-aware global placement followed by the default FastPlace-DP detailed placer.

- SAPTgp + SAPTdp: Our structure-aware global and detailed placement techniques.

We compare against the following state-of-the-art academic placers: Capo v10.2 [121], mPL6 [26], NTUPlace3 v7.10.19 [36], Rooster [123], FastPlace v3.0 [140], Dragon v3.01 [137] and SimPL [86]. All placers were supplied a target density of 1.0 as defined in the ISPD 2005 placement contest [107]. Finally, all HPWL and StWL results are reported on legalized placements using the coalesCgrip [41] tool.

7.6.1 Benchmark circuits

Table 7.4 provides the benchmark circuit characteristics. Of note is the number of alignment groups g_k and the datapath ratio in each design, where the datapath ratio is defined as the ratio of alignment group cells to random-logic cells. Though the hybrid designs are on the smaller side, they are state-of-the-art industrial circuits, and pose challenges for designers as they contain regular structures intermixed with other random logic. This makes it difficult to place the datapath logic separately from the random logic. The ISPD 2011 Datapath Benchmark Suite contains two datapath circuits, each with eight different utilizations to examine the ability of automatic placers to generate placement solutions at different densities on highly regular structures. In this work, the benchmarks were modified to make all the latches movable compared to the fixed latch placement in the original work. All logical connections and I/O pin locations remain the same. We compare against the modified datapath benchmarks as *unfixed latch* placement is more challenging and often indicative of a hybrid industrial design.

	ISPD 2011 Datapath Benchmarks		Industrial Hybrid Designs					
	A	B	C	D	E	F	G	H
# of nodes	160416	152668	17922	55387	83802	263906	194271	62133
# of pins	637984	653116	64078	94682	130000	397652	343727	21124
# of nets	157849	148682	16874	14458	16422	53884	62145	101582
# of g_k	1425	1932	35	110	60	131	82	28
Datapath ratio	0.920	0.850	0.010	0.012	0.008	0.007	0.018	0.021

Table 7.4: Circuit statistics. Datapath ratio is calculated as the total number of datapath cells divided by the total number of cells.

Design	ISPD 2011 Datapath Benchmark A: Total HPWL								ISPD 2011 Datapath Benchmark B: Total HPWL							
Util.	94	91	89	86	84	82	79	77	95	93	91	89	86	84	81	79
Capo	1.05	1.04	1.04	1.04	1.03	1.02	1.06	1.03	1.20	1.18	1.17	1.12	1.13	1.13	1.14	1.12
mPL6	1.17	1.19	1.22	1.14	1.16	1.20	1.17	1.16	1.64	1.86	1.72	1.64	1.65	1.65	1.78	1.78
NTUPl3	1.22	1.19	1.16	1.19	1.15	1.19	1.23	1.26	1.25	1.19	1.17	1.15	1.16	1.15	1.12	1.15
Dragon	1.49	1.58	1.63	1.60	1.51	1.62	1.66	1.60	1.40	1.40	1.35	1.32	1.32	1.30	1.31	1.31
FastPl3	1.42	1.50	1.53	1.54	1.53	1.67	1.70	1.75	1.69	1.66	1.73	1.71	1.77	1.86	1.77	1.87
Rooster	1.05	1.04	1.04	1.04	1.03	1.02	1.06	1.03	1.15	1.15	1.13	1.12	1.15	1.13	1.13	1.13
SimPL	1.08	1.07	1.06	1.07	1.05	1.06	1.05	1.04	1.23	1.22	1.21	1.20	1.17	1.16	1.16	1.15
SAPTgp	1.07	1.04	1.05	1.03	1.03	1.02	1.02	1.01	1.21	1.20	1.17	1.16	1.16	1.16	1.16	1.15
SAPTdp	1.06	0.99	1.03	1.02	1.02	1.01	1.01	0.98	1.21	1.19	1.17	1.16	1.15	1.15	1.14	1.15

Table 7.5: Total HPWL ratio comparison on the modified ISPD 2011 Datapath Benchmark A and B variants with legalized placement. The ratios are computed with respect to the manually placed solution.

7.6.2 Wirelength results on ISPD2011 datapath benchmarks

Tables 7.5 and 7.6 compare the total HPWL and total StWL on the ISPD2011 Datapath Benchmark Suite. To clarify, the total wirelength calculation includes both random-logic and datapath nets in the design. All results are the ratio of the total wirelength of the automatically placed solution to the total wirelength of the manually designed placement as described in [149]. From Table 7.5 we observe that even though we apply specialized techniques to align the datapath logic, the total HPWL of our placement solution is within a few percentage points of the best HPWL solution for each of the benchmark variants. In fact, we obtain the best HPWL on seven out of eight variants for benchmark A.

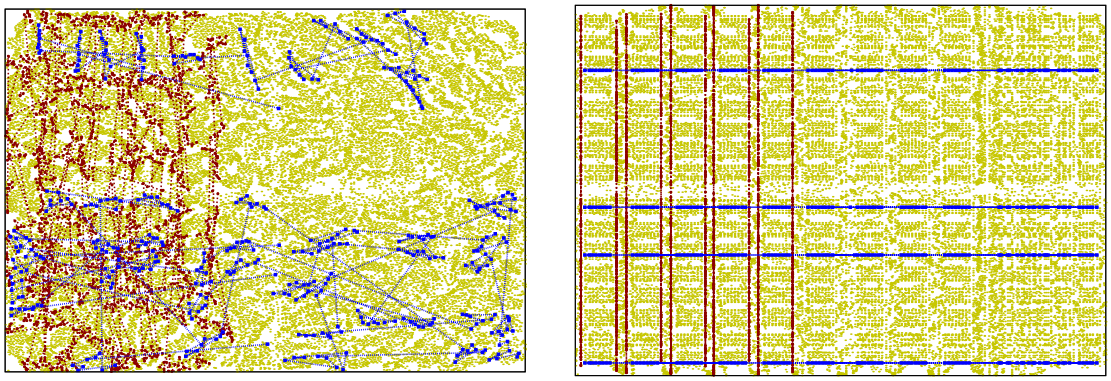
As demonstrated in Section 7.2, HPWL is a poor indicator of the datapath placement

Design	ISPD 2011 Datapath Benchmark A: Total StWL								ISPD 2011 Datapath Benchmark B: Total StWL							
Util.	94	91	89	86	84	82	79	77	95	93	91	89	86	84	81	79
Capo	1.94	1.94	1.91	1.93	1.90	1.90	1.80	1.90	2.40	2.40	2.38	2.35	2.36	2.36	2.35	2.32
mPL6	2.16	2.14	2.16	2.08	2.10	2.12	2.11	2.09	2.94	3.29	3.06	3.01	2.97	2.95	3.20	3.21
NTUPl3	2.23	2.18	2.15	2.15	2.11	2.16	2.19	2.09	2.66	2.48	2.47	2.44	2.44	2.44	2.32	2.44
Dragon	2.37	2.44	2.53	2.48	2.36	2.48	2.56	2.43	2.91	2.87	2.84	2.80	2.79	2.77	2.75	2.74
FastPl3	2.45	2.53	2.56	2.59	2.56	2.71	2.75	2.79	3.73	3.58	3.78	3.79	3.97	4.13	3.96	4.14
Rooster	1.94	1.93	1.91	1.91	1.92	1.91	1.82	1.95	2.40	2.39	2.35	2.33	2.38	2.34	2.34	2.32
SimPL	1.82	1.83	1.80	1.81	1.78	1.78	1.78	1.75	2.27	2.30	2.25	2.24	2.23	2.19	2.24	2.22
SAPTgp	1.40	1.35	1.36	1.33	1.33	1.32	1.31	1.31	1.59	1.56	1.54	1.50	1.51	1.50	1.50	1.48
SAPTdp	1.35	1.28	1.31	1.31	1.28	1.38	**1.26**	1.28	1.58	1.55	1.52	1.49	1.49	1.48	1.48	**1.46**

Table 7.6: Total StWL ratio comparison on the modified ISPD 2011 Datapath Benchmark A and B variants with *unfixed latches* after legalized placement. The ratios are computed with respect to the manually placed solution. Numbers in bold are the best automated placement results published for these benchmarks.

quality, especially in the case of highly structured designs, and StWL is much more accurate. Additionally, StWL correlates well with design routability. From Table 7.6 we observe that our proposed techniques (SAPTgp + SAPTdp) obtained significantly better StWL among all the automated solutions for all benchmark variants. For the benchmark A variants, StWL of all the other placers is greater than 1.5 times the manually designed solution. For the benchmark B variants, the results are greater than 2.0 times the manually designed solution. Alternatively, for benchmark A, the proposed global placement techniques (SAPTgp + FP-DP) are able to obtain a solution that is at 1.31 times the manually designed solution (at 79% utilization). Additionally, the detailed placement techniques (SAPTgp + SAPTdp) are able to further reduce the gap to 1.26 times. The proposed placer on benchmark B also significantly outperformed prior placers, with SAPTgp + FP-DP achieving 1.48 times and SAPTgp + SAPTdp achieving 1.46 times StWL of the manually designed solution.

Figure 7.8a displays the datapath placement solution from SimPL. In this figure, a random selection of cells in alignment groups are plotted with a brown or blue line connecting



(a) (b)

Figure 7.8: Forty structured bit-stacks are randomly chosen to show the alignment impact of the proposed placer in Benchmark A. (a) is generated by SimPL [86] whereas (b) is generated by the proposed placer. Movable cells are shown lightly shaded while the cells in the datapath group are shown dark. Note that cells that are not defined by alignment groups become aligned as well and form a regular structure (right).

them. In the manually placed solution, these cells are aligned, either vertically or horizontally depending on the group, yielding shorter wirelength. From Figure 7.8a, clearly the cells are not placed aligned. The placement solution generated using the proposed placer is shown in Figure 7.8b with the same set of alignment groups highlighted as Figure 7.8a. Clearly there is significant improvement in the alignment of each g_k.

7.6.3 Wirelength results on hybrid designs

Table 7.7 gives the normalized total HPWL and StWL results on the hybrid designs that integrate datapaths into larger netlists. All results are normalized to the proposed placer (SAPTgp + SAPTdp). As before, the proposed placer generates significantly better StWL on the hybrid designs. Though the HPWL results are similar, the proposed placer obtains an improvement in StWL between 1% and 13%. For these designs, improvement in StWL is significant, considering the fact that the percentage of datapath logics within

the designs is less than 3%.

By providing alignment constraints to portions of the datapath, we observe that neighboring cells also become aligned during the iterative placement process (Figure 7.8). The alignment constraints provide hints, directing the placer in the correct gradient. These hints help to overcome local optima, driving placement towards a more globally optimal solution. Thus, with relatively few manually pre-defined alignment groups, this work shows that a HPWL-driven placer can generate improved solutions for the other cells, resulting in significantly improved StWL.

7.6.4 Routing congestion results

To empirically prove our claim that StWL accurately approximates routability, we report results using two congestion metrics on all benchmark circuits. It is important to note that the proposed placer is not doing any congestion-aware placement techniques, and that any routing gains are simply a by-product of improved placement solutions.

Table 7.8 displays the total overflow (TOF)($\times 1e+5$), as defined in the ISPD 2011 routability-driven placement contest [143] for the datapath benchmark circuits. All re-

	Hybrid C		Hybrid D		Hybrid E		Hybrid F		Hybrid G		Hybrid H	
	HPWL	StWL	HPWL	StWL	HPWL	StWL	HPWL	StWL	HPWL	StWL	HPWL	StWL
Capo	1.13	1.26	1.17	1.32	1.12	1.27	1.19	1.17	1.25	1.23	1.23	1.21
mPL6	1.05	1.15	1.02	1.14	1.20	1.32	1.37	1.30	1.21	1.19	1.09	1.10
NTUPl3	0.95	1.10	0.95	1.13	0.99	1.19	1.30	1.30	1.02	1.05	1.00	1.01
Dragon	1.10	1.20	2.11	2.04	1.32	1.38	1.29	1.24	-	-	1.22	1.20
Rooster	1.10	1.19	1.16	1.32	1.26	1.23	1.03	1.25	1.03	1.05	1.26	1.23
FastPl3	0.95	1.04	0.96	1.16	1.22	1.30	1.17	1.14	1.11	1.10	1.04	1.03
SimPL	1.02	1.10	0.97	1.16	1.03	1.12	1.04	1.04	1.01	1.01	1.02	1.02
SAPT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 7.7: Total HPWL and StWL ratio comparison on hybrid designs. The wirelength ratios are compared to the proposed placer. The Dragon placer was unable to complete for Hybrid G.

Design	ISPD 2011 Datapath Benchmark A: Routing Overflow								ISPD 2011 Datapath Benchmark B: Routing Overflow							
Util.	94	91	89	86	84	82	79	77	95	93	91	89	86	84	81	79
Capo	2.29	2.17	1.72	1.83	1.84	1.68	1.10	2.18	9.16	7.28	7.05	6.68	7.17	7.01	7.13	6.98
mPL6	4.66	4.38	4.44	3.40	3.38	3.65	6.03	5.02	12.7	16.4	14.0	13.6	12.8	12.6	15.3	15.3
NTUPl3	5.54	5.12	4.63	5.19	4.92	5.63	6.03	5.02	10.2	8.41	8.3	8.09	8.92	9.07	8.21	9.92
Rooster	0.79	1.08	1.22	1.02	1.07	1.64	1.31	1.16	8.11	7.28	6.88	6.72	-	-	-	-
FastPl3	7.23	8.10	8.72	9.08	8.80	10.4	11.8	12.1	20.8	19.3	21.6	21.7	23.7	25.5	23.5	25.6
SimPL	1.28	1.28	1.22	0.98	0.87	0.87	0.85	0.77	5.98	6.24	5.65	5.49	5.26	4.85	5.21	5.25
SAPTgp	0.0012	0.032	0.0	0.0	0.0	0.0	0.0	0.0	0.90	0.70	0.56	0.45	0.48	0.59	0.62	0.59
SAPTdp	0.0014	0.038	0.0	0.0	0.0	0.0	0.0	0.0	0.88	0.70	0.55	0.43	0.67	0.58	0.60	0.58

Table 7.8: Total Overflow ($\times$10e+5) results produced with the router and evaluation script from the ISPD 2011 routability-driven placement contest on the modified ISPD 2011 Datapath Benchmark A and B variants with *unfixed latches* after legalized placement. The "Total Overflow", a measure of the routing congestion of the placement solution, is reduced to zero on six of the benchmark A variants and reduced by at least 6.7 times for all benchmark B variants.

Design	ISPD 2011 Datapath Benchmark A: Average PWC								ISPD 2011 Datapath Benchmark B: Average PWC							
Util.	94	91	89	86	84	82	79	77	95	93	91	89	86	84	81	79
Capo	123.4	123.5	121.3	121.1	121.4	120.3	118.5	128.4	237.4	222.4	222.5	217.4	222.3	220.1	224.3	219.6
mPL6	170.2	188.2	187.3	149.1	145.3	159.9	149.6	139.9	278.2	325.2	313.2	300.0	275.0	315.9	302.9	361.4
NTUPl3	171.1	170.3	152.1	170.8	166.7	176.9	189.9	173.4	262.5	237.4	239.4	241.4	247.2	247.1	238.6	259.2
Rooster	136.6	163.0	156.5	147.4	152.8	169.4	162.7	174.2	256.2	224.1	217.3	215.1	231.5	221.8	228.7	225.2
FastPl3	211.9	236.1	237.3	234.7	222.2	224.7	240.2	284.9	341.8	316.2	331.6	334.1	397.6	347.0	341.4	341.7
SimPL	124.7	118.8	117.8	116.2	115.1	114.8	114.1	119.5	207.8	211.3	205.2	205.0	212.5	209.3	204.6	209.2
SAPTgp	100.6	100.2	100.1	100.1	100.1	100.1	100.1	100.1	152.0	150.7	149.0	148.6	148.2	148.3	149.4	150.1
SAPTdp	100.4	100.1	100.1	100.1	100.1	100.1	100.1	100.1	151.9	150.7	149.0	148.5	147.9	148.1	149.2	150.1

Table 7.9: Average Peak Weighted Congestion (PWC) in datapath benchmark.

Routing	Hybrid C		Hybrid D		Hybrid E		Hybrid F		Hybrid G		Hybrid H	
Metrics	TOF	PWC	TOF	PWC	TOF	PWC	TOF	PWC	TOF	PWC	TOF	PWC
Capo	1458	107.34	3024	110.46	26504	117.29	41520	114.73	131543	155.56	1186	115.52
mPL6	1360	106.90	940	101.88	23838	117.02	17801	111.17	45904	122.08	490	106.51
NTUPl3	626	103.66	558	95.92	32545	133.99	114843	141.76	42917	121.33	98	101.36
Rooster	48	100.38	1419	105.22	32157	122.32	61731	118.80	134434	149.26	918	111.49
FastPl3	372	102.25	722	99.52	77774	165.88	34189	118.21	33723	115.24	30	100.40
SimPL	650	103.31	475	94.60	19713	118.73	29524	113.67	37884	120.33	100	101.11
SAPT	534	103.49	322	93.21	11276	114.82	28736	113.66	40275	119.12	28	100.34

Table 7.10: Total Overflow (TOF) and ACE routing metrics for hybrid designs.

ported overflow values are from the official contest evaluation script. As seen in Table 7.8, SAPT produces the smallest overflow across all testcases. For benchmark A, SAPT produced a routable placement solution with *zero* (0) overflow for all but two of the vari-

ations. For benchmark B, SAPT improves total overflow by 6.7 times, 23.54 times, 14.44 times, 11.56 times, 14.3 times, and 10.36 times versus SimPL, FastPlace3.0, Dragon, NTUPlace3, mPL6, and Capo respectively. The second congestion metric is the Peak Weighted Congestion (PWC) as used in the DAC 2012 routability-driven placement contest [144]. This metric calculates the weighted average ACE(x) congestion [153], of the the top $x\%$ congested g-edges. The smaller the PWC value, the more routable the design. As with the overflow metric, in designs with significant structure, the SAPT techniques produce significantly more routable designs than the compared placers.

Table 7.10 displays the TOF using the ISPD 2011 contest router (coalesCgrip [41]) and the PWC using the DAC 2012 contest router (BFG-R [67]) for the hybrid designs. SAPT generated the best PWC results for Hybrids D, E, and G, and was comparable to the best results on the remaining designs. Additionally, SAPT generated the best TOF solution for three Hybrid designs D, E, and H. Though SAPT is not a congestion-aware placer, the significant improvement in routing congestion indicates the strong correlation between alignment and congestion.

7.6.5 Runtime results

Runtime results on the Hybrid designs are shown in Table 7.11. Both SimPL and FastPlace3.0 were similar with FastPlace3.0 quicker on larger designs. Our SAPT placer performed very competitively compared to the other state-of-the-art placement flows with the largest design, Hybrid F, taking under 71 seconds to place.

Hybrid	C	D	E	F	G	H
Capo	94.6	74.0	83.4	480.3	482.7	65.9
mPL6	48.5	32.4	36.2	161.7	148.5	35.9
NTUPl3	13.0	30.0	70.0	278.0	269.0	54.0
Dragon	425.9	193.0	283.9	927.4	-	154.9
Rooster	98.8	84.1	101.1	560.7	455.1	80.9
FastPl3	13.0	10.7	17.4	55.3	32.93	7.2
SimPL	9.2	12.6	27.1	59.2	40.4	9.2
SAPT	15.9	16.7	38.2	70.9	43.1	10.8

Table 7.11: Comparison of runtime on the hybrid designs.

7.7 Summary and Conclusions

This chapter presents a unified framework to enhance current random-logic placers to better handle designs containing datapath logic. A set of new global placement techniques, including skewed weighting with step size scheduling, fixed-point and pseudo net alignment constraint, bit-stack aligned cell-swapping and group recursive repartitioning, were presented that seamlessly integrate alignment constraints into a state-of-the-art placement engine to overcome the shortcomings of the HPWL model for datapaths. Experimental results show at least a 28% improvement in total StWL compared with six state-of-the-art academic placers for the ISPD 2011 Datapath Benchmark Suite and a 5.8% average improvement in total StWL over six state-of-the-art placers for industrial hybrid designs. Though comparisons do not report the timing impact because the current implementation is limited to reading the Bookshelf format, significant improvements in wirelength are generally attributed to improved timing. An open challenge is to quantify this effect in addition to automatically extracting the datapath.

CHAPTER VIII

Conclusions

This chapter summarizes the results of our research and outlines open challenges.

8.1 Our Results in Perspective

In addition to improving quality and speed of core circuit optimization algorithms, the research presented in this dissertation pursues several recurring themes in physical design and physical synthesis. **One** is *the comparisons and tradeoffs between linear and quadratic wirelength functions*. Since the 1960s, it was known that quadratic optimization was computationally efficient, but did not adequately track the demand for routing resources, which is much closer to the HPWL objective and its weighted variants [22]. Seminal work by Doll and Johannes in the early 1990s developed a *linearization* technique that represents the linear wirelength objective on graphs by a dynamically-weighted quadratic objective [132]. However, the modeling of multi-pin nets remained inaccurate, and the research community has largely replaced quadratic optimization by much more cumbersome and slow nonlinear optimization techniques ten years later [26, 36, 79]. In the mid-2000s, Spindler and Johannes developed the Bound2Bound model [134], which considerably improved the modeling accuracy for multi-pin nets in quadratic placement

by employing a dynamic (placement-dependent) graph topology. With additional improvements to flat quadratic placement, this technique has recently outperformed prior art in both runtime and quality of results, both in terms of HPWL and in routability-driven placement [57, 85, 86]. This development raised several key research questions:

- Is there a gap between the Bound2Bound model and the HPWL in practice?
- Can global quadratic optimization with the Bound2Bound model be effectively improved on multi-million gate netlists (with respect to HPWL)?
- Is multilevel placement optimization compatible with Bound2Bound and competitive in performance?

Our work answers these three questions in the affirmative. The gap between Bound2Bound and HPWL is illustrated by the SimPL line in Figure 5.3 — note the *return* to smaller HPWL when detailed placement is invoked. Global quadratic placement of multi-million gate netlists can be improved by using the ProLR technique proposed in Section 5.5. MAPLE (Chapter V) demonstrates that multilevel placement techniques are compatible with the Bound2Bound model and is competitive with state of the art, as long as abrupt changes to placement are avoided before/after clustering. However, Section 5.7.3 shows that only two levels of clustering are useful for current benchmarks. Larger netlists may justify deeper clustering.

The second theme addressed in this dissertation is relatively new to physical design, but no less fundamental — *methodology for module spreading and handling of whitespace*. These considerations are essential not only to global placement, but also buffer insertion, gate sizing and other physical synthesis transformations, as well as to congestion-driven

placement. Until the late 1990s, whitespace was rare in IC layouts, but now can reach over 60% by area [107]. We develop efficient techniques for spreading modules during placement, while satisfying density constraints and optimizing HPWL beyond the accuracy of the Bound2Bound model (Chapters IV, V and VI).

The third fundamental theme explored in this dissertation has not received as much recognition, but may deserve it — we study *the composition of multiple optimizations into a high-precision, reliable multiobjective optimization process*. Our key discovery is that transitions between multiple objective functions and optimization techniques in placement often lead to major disruptions. In particular, adding netlist clustering or Iterative Local Refinement (ILR) [140] to the SimPL algorithm for quadratic placement with the Bound2Bound model does not directly improve quality of results because the disruptions overshadow the benefits of such integration. To this end, we develop new techniques, such as two-tier Progressive Local Refinement (ProLR) described in Chapter V, to facilitate graceful transitions between multiple optimizations. In placement, these techniques are applied before and after unclustering, during the transition from a quadratic objective to HPWL, and before detailed placement. Other applications exist in physical synthesis.

The fourth theme is *early consideration of routability*. Routability has not been a critical issue until recently. However, with ever-increasing chip complexity and growing amount of global wiring, even ten metal routing layers are often insufficient to accommodate demand for routing resources. A great amount of effort is employed in physical design to resolve routing congestion, and traditional wirelength-driven optimization are being extended to consider routability. IBM Research organized two routability-driven placement

contests in the past two years. To this end, our SIMultaneous PLace-and-Route algorithm SimPLR (Chapter IV) integrates a layer-aware global routing into SimPL (Chapter III), giving the placer early access to actual routing congestion maps. Compared to the state of the art in wirelength-driven placement, this new technique reduces routing congestion by 4.5 times.

The fifth theme studied in this dissertation — *datapath layout* — is a long-lasting challenge in physical design. Many attempts have been made in the last 40 years to close the quality gap between manual and automated placement of datapaths and other regular structures. Nevertheless, a common assumption still prevails among IC designers that circuits with high regularity require manual placement. Our key observation is that the primary optimization objective of modern state-of-the-art placement algorithms – HPWL can mislead placers on datapath-oriented designs. In particular, compressing placement of high-fanout nets to lower the overall HPWL can disrupt the regularity of placement and undermine its Steiner wirelength, which is known to better correlate with routed wirelength. Based on this observation, we develop a suite of structure-aware placement techniques, SAPT (Chapter VII) that can guide existing HPWL-driven placers to carefully handle these nets, enhancing layout regularity and Steiner wirelength.

8.2 The Impact of Our Research

Our research has been reported at top conferences in Electronic Design Automation and Physical Design — DAC, ICCAD, and ISPD — and published in IEEE Transactions on Computer-Aided Design of Integrated Circuits. Since its first publication at ICCAD 2010, the SimPL algorithm saw rapid adoption in both academia and industry, as sum-

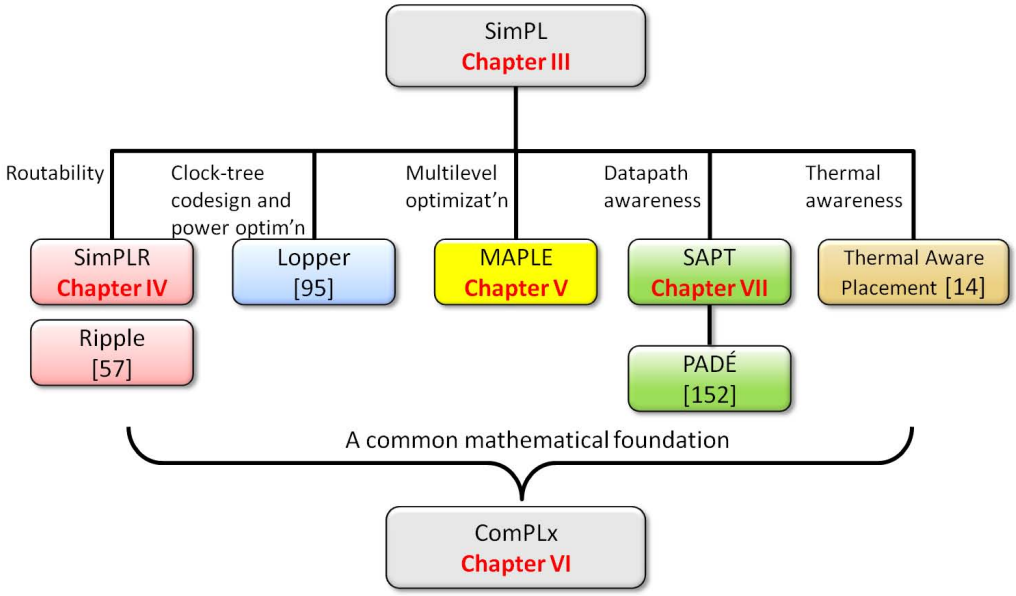


Figure 8.1: The SimPL family of placement algorithms. The baseline algorithm has been re-implemented by four research groups, facilitating further extensions.

marized in Figure 8.1. Our approach consistently outperforms previous state of the art in speed and solution quality, is amenable to thread-level and instruction-level parallelism, requires only a modest amount of code, and was successfully re-implemented by independent researchers [14, 57]. Our approach also supports a variety of discrete and continuous constraints. It is now extended to power-aware placement with integrated clock-network synthesis (Lopper) [94,95] and thermal-aware placement [14], as well as routability-driven placement (Ripple and SimPLR). Ripple [57] and our software SimPLR (Chapter IV) finished in top three at the ISPD 2011 and DAC 2012 routability-driven placement contest. In joint work with IBM Research, the SimPL algorithm has been extended to multilevel optimization (MAPLE) and structure-aware placement (SAPT). Further extensions of SAPT combined it with datapath extraction techniques (PADE) [152]. As of August 2012, MAPLE is the default placement algorithm in IBM's Placement-Driven-Synthesis flow for both ASICs and CPUs.

8.3 Open Challenges

Several open challenges are suggested by our results. One challenge is combining quadratic force-directed placement algorithms with placement algorithms based on nonconvex optimization. Empirical results in Tables 5.3, 5.4, 6.1, and 6.2 indicate a trend — quadratic placers RQL, SimPL and MAPLE produce comparable solutions than placers APlace3, NTUPlace3 and mPL6 based on nonconvex optimization, but significantly reduce runtimes. This trend is confirmed in the results of the ISPD 2011 routability-driven placement contests and subsequent work [57, 85] — quadratic placers Ripple and SimPLR outperformed nonconvex placers mPL6 and NTUPlace3. This is due, in part, to the greater amount of recent research on quadratic placement, including the development of successful industry tools [18, 141]. Yet, many of our contributions, such as lookahead legalization (LAL) and ProLR, can be adapted for use in nonconvex placers. Conceivably, the two types of placers can be combined to leverage the best qualities of each.

Tighter integration of timing optimizations into placement algorithms is also a possibility. In Chapter VI, our ComPLx work demonstrated that net weights can be used to effectively reduce the length of specific nets without undermining overall solution quality. However, given that timing-critical nets are typically identified by sign-off quality timing engines after placement, significant placement modification can be required in presence of a large number of near-timing critical nets. This placement modification can undermine placement quality, generate new critical nets, and hamper timing closure. New fast and accurate timing evaluators and their integration with placement can reduce effort for timing closure in modern design flows.

In response to technology and business trends, the number of macros included in modern chips is growing [154]. Handling these macros during physical design will require an increasing amount of effort in the near future. While research on floorplanning has a long history (dating back to the 1980s), this research has intensified in the early 2000s when Andrew Kahng formulated the concept of *fixed-outline floorplanning* in [75]. Starting at about the same time [2, 50], macro placement has been combined with standard-cell placement. Numerous analytical and combinatorial techniques for such *mixed-size placement* have been developed. However, the problems of determining locations of macros and standard cells are essentially the same from the optimization point of view, distinguished only by (i) the scale relative to the size of layout regions, and (ii) shaping and rotations of macros in floorplanning. In principle, a single EDA tool that simultaneously places macros and standard cells can better optimize the overall placement. Given strong performance of analytical placement algorithms and poor scalability of combinatorial floorplanning, we expect that a unified analytical approach to mixed-size placement will eventually dominate. In Chapter VI, we demonstrated our one-stage approach to handle macros in a unified analytical placement framework without support from combinatorial floorplanning. Further extensions of this approach are currently impeded by the algorithmic challenge of quickly removing overlaps between macros without significant interconnect overhead [26, 36, 79, 105, 140], especially in the presence of large macros and/or high design utilization. In particular, analytical placement algorithms for global placement temporarily allow overlaps during their iterations (in order to focus on interconnect optimization), but are unable to clear the resulting overlaps entirely. These algorithms also approximate or

disregard the shapes of movable objects. To ensure that post-global placement is close to legal, it is important to control overlap during iterations and also to account for the shapes and rotations of movable cells.

Another direction where we foresee further studies is the automatic generation of datapath layout. In Chapter VII, our work SAPT identified the pitfalls of modern HPWL-driven current placement algorithms and demonstrated the significant improvement in Steiner wirelength over the prior state of the art. However, we also pointed out that a non-trivial quality gap still remains compared to the manual placement. This suboptimality of automatic placement requires accurate extraction of datapath logic and optimal spacing between alignment groups, as well as structure-aware legalization algorithms.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] S. N. Adya and I. L. Markov, "Executable Placement Utilities", `http://vlsicad.eecs.umich.edu/BK/PlaceUtils/`

[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, "Combinatorial techniques for Mixed-size Placement", *ACM Trans. Design Autom. Electr. Syst.* 10(1), 2005, pp. 58-90.

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

[5] A. Agnihotri et al, "Mixed Block Placement via Fractional Cut Recursive Bisection", *IEEE TCAD* 24(5), pp. 748-761, 2003.

[6] R. K. Ahuja, T. L. Magnati, J. B. Orlin, "Network Flows: Theory, Algorithms, and Applications," *Prentice Hall* 1993.

[7] C. J. Alpert, T. F. Chan, A. B. Kahng, I. L. Markov and P. Mulet, "Faster Minimization of Linear Wirelength for Global Placement", *IEEE TCAD* 17(1) 1998, pp. 3-13, 1998.

[8] C. J. Alpert, G.-J. Nam and P. G. Villarrubia, "Effective Free Space management for Cut-based Placement via Analytical Constraint Generation", *IEEE TCAD* 22(10), pp. 1343-1353, 2003.

[9] C. J. Alpert et al., "A Semi-persistent Clustering Technique for VLSI Circuit Placement", *Proc. ISPD*, pp. 200-207, 2005.

[10] C. J. Alpert et al., "Techniques for Fast Physical Synthesis", *Proc. IEEE* 95(3), 2007, pp. 573-599.

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

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

[13] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", *Proc. AFIPS* (30), pp. 483-485, 1967.

[14] S. Aroonsatidecha et al, "A Fast Thermal Aware Placement with Accurate Thermal Analysis Based on Green Function", *Prof. ASP-DAC* 2012, pp. 425-430.

[15] D. P. Bertsekas, "Nonlinear Programming", 2nd ed., *Athena Scientific* 1999.

[16] S. Boyd, L. Xiao and A. Mutapcic, "Subgradient Methods", Notes for EE392o, *Stanford University* 2003. `http://www.stanford.edu/class/ee392o/subgrad_method.pdf`

[17] S. Boyd and L. Vandenberghe, "Convex Optimization", *Cambridge University Press*, 2004. `http://www.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`

[18] U. Brenner, M. Struzyna and J. Vygen, "BonnPlace: Placement of Leading-Edge Chips by Advanced Combinatorial Algorithms", *IEEE TCAD* 27(9) 2008, pp.1607-20.

[19] M. Breuer, "Min-cut Placement", *Journal of Design Automation and Fault Tolerant Computing*, 1(4) 1977, pp.343-362.

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

[21] N. Buchbinder and J. Naor, *The Design of Competitive Online Algorithms via a Primal-Dual Approach*, NOW Publishers, 2009.

[22] A. E. Caldwell et al., "On Wirelength Estimations for Row-based Placement", *IEEE TCAD* 18(9), 1999, pp. 1265-1278.

[23] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?" *Proc. DAC* 2000.

[24] T. F. Chan, J. Cong and E. Radke, "A Rigorous Framework for Convergent Net-weighting Schemes in Timing-driven Placement", *Proc. ICCAD*, pp. 288-294, 2009.

[25] T. F. Chan, J. Cong and K. Sze, "Multilevel Generalized Force-directed Method for Circuit Placement", *Proc. ISPD*, pp. 185-192, 2005.

[26] T. F. Chan et al., "mPL6: Enhanced Multilevel Mixed-Size Placement", *Proc. ISPD*, pp. 212-214, 2006.

[27] C. C. Chang et al., "A Practical All-Path Timing-driven Place and Route Sytem", *Proc. ASP-DAC*, pp. 560-563, 1994.

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

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

[30] D. Chai and A. Kuehlmann, "Efficient Symmtry-baed Circuit Rewiring for Placement Optimization", *IWLS*, 2008.

[31] H. Chen et al., "An Algebraic Multigrid Solver for Analytical Placement with Layout Based Clustering", *Proc. DAC*, pp. 794-799, 2003.

[32] H.-C. Chen et al., "Constraint Graph-based Macro Placement for Modern Mixed-size Circuit Designs", *Proc. ICCAD*, pp. 218-223, 2008.

[33] T.-C. Chen et al.,"MP-trees: A Packing-based Macro Placement Algorithm for Mixed-size Designs", *IEEE TCAD* 27(9) 2008.

[34] T.-C. Chen, M. Cho, D. Z. Pan and Y.-W. Chang, "Metal-Density-Drivn Placement for CMP Variation and Routability", *IEEE TCAD* 27(12) 2008.

[35] T.-C. Chen, A. Chakraborty and D. Z. Pan, "An Integrated Nonlinear Placement Framework with Congestion and Porosity Aware Buffer Planning", *Proc. DAC* 2008.

[36] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen and Y.-W. Chang, "NTUPlace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints", *IEEE TCAD* 27(7) 2008, pp.1228-1240.

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

[38] A. Chowdhary et al. "Extraction of Functional Regularity in Datapath Circuits", *IEEE TCAD* 18(9), pp. 1279-1296, 1999.

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

[40] Y.-L. Chuang et al., "Design-hierarchy Aware Mixed-size Placement for Routability Optimization", *Proc. ICCAD*, pp.663-668, 2010.

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

[42] J. Cong, M. Romesis and J. Shinnerl, "Robust Mixed-Size Placement Under Tight White-Space Constraints," *ICCAD* 2005, pp. 165-172.

[43] J. Cong, M. Romesis and J. R. Shinnerl, "Fast Floorplanning by Look-ahead Enabled Recursive Bipartitioning", *IEEE TCAD* 25(9), pp. 1719-1732, 2006.

[44] J. Cong et al., "Thermal-Aware 3D IC Placement Via Transformation", *Proc. ASP-DAC* pp. 780-785, 2007.

[45] J. Cong, M. Xie, "A Robust Mixed-Size Legalization and Detailed Placement Algorithm", *IEEE TCAD* 27(8), pp. 1349-1362, 2008.

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

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

[48] K. Doll, F. M. Johannes and K. J. Antreich, "Iterative Placement Improvement By Network Flow Methods", *IEEE TCAD* 13(10), pp. 1189-1200, 1994.

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

[50] H. Eisenmann, F. M. Johannes, "Generic Global Placement and Floorplanning", *DAC*, pp. 269-274, 1998.

[51] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", *DAC*, pp. 175-181, 1982.

[52] M. Garland, "Sparse Matrix Computations on Manycore GPU's", *Proc. DAC*, pp. 2-6, 2010.

[53] P. E. Gill and D. P. Robinson, "A Primal-Dual Augmented Lagrangian", *Computational Optimization and Applications* 2010, DOI: 10.1007/s10589-010-9339-1.

[54] S. Goto, "An Efficient Algorithm for the Two-Dimensional Placement Problem in Electrical Circuit Layout", *Trans. on Circuits and Systems* 28(1), pp. 12-18, 1981.

[55] G. Goumas et al., "Understanding the Performance of Sparse Matrix-Vector Multiplication", *Euromicro Int'l Conf. on PDP*, pp. 283-292, 2008.

[56] L. K. Grover, "A New Simulated Annelaing Algorithm for Standard Cell Placement", *Proc. ICCAD*, pp. 646-650, 1984.

[57] X. He, T. Huang, L. Xiao, H. Tian, G. Cui and E. F. Y Young, "Ripple: An Effective Routability-Driven Placer by Iterative Cell Movement", *Proc. ICCAD* 2011.

[58] R. Ho, K. W. Mai and M. A. Horowitz, "The Future of Wires", Proceedings of the IEEE 89(4), pp. 490-504, 2001.

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

[60] L. Hsu et al., "Exploring the Cache Design Space for Large Scale CMPs", *ACM SIGARCH Computer Architecture News*, pp. 24-33, 2005.

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

[62] M.-K. Hsu and Y.-W. Chang, "Unified Analytical Global Placement for Large-scale Mixed-size Circuit Designs", *Proc. ICCAD*, pp. 657-662, 2010.

[63] M.-K. Hsu, Y.-W. Chang and V. Balabanov, "TSV-aware Analytical Placement for 3D IC Designs", *Proc. DAC*, pp. 664-669, 2011.

[64] M.-K. Hsu et al., "Routability-Driven Analytical Placement for Mixed-Size Circuit Designs", *Proc. ICCAD*, pp. 80-84, 2001.

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

[66] B. Hu and M. Marek-Sadowska, "mFAR: Fixed-Points-Addition-based VLSI Placement Algorithm", *Proc. ISPD*, pp. 239-241, 2005.

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

[68] P. Ienne and A. Griebing, "Practical Experiences with Standard-Cell Based Datapath Design Tools", *Proc. DAC*, pp. 396-401, 1998.

[69] ISPD 2007 Global Routing Contest. `http://www.sigda.org/ispd2007/contest.html`.

[70] M. Jackson and E. S. Kuh, "Performance-Driven Placement of Cell Based IC's", *Proc. DAC*, pp. 370-385, 1989.

[71] D. A. Jamsek, "Designing and Optimizing Compute Kernels on NVIDIV GPUs", *Proc. ASP-DAC*, pp. 224-229, 2009.

[72] Devang Jariwala and John Lillis, "RBI: Simultaneous Placement and Routing Optimization Technique", *IEEE TCAD* 26(1), pp. 127-141, 2007.

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

[74] M. Jones and P. Banerjee, "An Improved Simulated Annelaing Algorithm for Standard Cell Placement", *Proc. ICCAD*, pp. 83-86, 1987.

[75] A. B. Kahng, "Classical Floorplanning Harmful?", *Proc. ISPD*, pp. 207-213, 2000.

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

[77] A. B. Kahng and Q. Wang, "An Analytic Placer for Mixed-size Placement and Timing-driven Placement", *Proc. ICCAD*, pp. 565-572, 2004.

[78] A. B. Kahng and Q. Wang, "Implementation and Extensibility of an Analytic Placer", *IEEE TCAD*, pp. 734-747, 2005.

[79] A. B. Kahng and Q. Wang, "A Faster Implementation of APlace", *Proc. ISPD*, pp. 218-220, 2006.

[80] A. B. Kahng, J. Lienig, I. L. Markov and J. Hu, "VLSI Physical Design: from Graph Partitioning to Timing Closure", Springer 2011, 312 pages.

[81] A. A. Kennings and I. L. Markov, "Smoothening Max-terms and Analytical Minimization of Half-Perimeter Wirelength", *VLSI Design* 14(3), pp. 229-237, 2002

[82] A. A. Kennings and K. Vorwerk, "Force-Directed Methods for Generic Placement", *IEEE TCAD* 25(10), 2006, pp. 2076-2087.

[83] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Systems Technical Journal* 49, pp. 291-307, 1970.

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

[85] M.-C. Kim, J. Hu, D.-J. Lee and I. L. Markov, "A SimPLR method for Routability-driven Placement", *Proc. ICCAD*, pp. 67-73, 2011.

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

[87] M.-C. Kim et al, "MAPLE: Multilevel Adaptive PLacEment for Mixed-Size Designs", *Proc. ISPD* 2012.

[88] M.-C. Kim and I. L. Markov, "ComPLx: A Competitive Primal-dual Lagrange Optimization", *Proc. DAC* 2012.

[89] K. C. Kiwiel, T. Larsson and P. O. Lindberg,"Lagrangian Relaxation via Ballstep Subgradient Methods",*Mathematics of Operations Research* 32(3), 2007, pp. 669-686. `http://mor.journal.informs.org/content/32/3/669`

[90] T. Kong, "A Novel Net Weighting Algorithm for Timing-driven Placement", *Proc. ICCAD*, pp. 172-176, 2002.

[91] T. Kutzschebauch and L. Stok, "Regularity Driven Logic Synthesis", *Proc. ICCAD*, pp. 439-446, 2000.

[92] J. J. Kleinhans et al., "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization", *IEEE TCAD* 10(3), 1991, pp. 356-365.

[93] D.-J. Lee, M.-C. Kim and I. L. Markov, "Low-Power Clock Trees for CPUs", *Proc. ICCAD*, pp. 444-451, 2010.

[94] D.-J. Lee and I. L. Markov, "Obstacle-Aware Clock-tree Shaping During Placement", *Proc. ISPD*, pp. 123-130, 2011.

[95] D.-J. Lee and I. L. Markov, "Obstacle-aware Clock-tree Shaping during Placement", *IEEE TCAD* 31(2), 2012.

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

[97] W.-P. Lee, H.-Y. Liu and Y.-W. Chang, "An ILP Algorithm for Post-floorplanning Voltage-Island Generation Considering Power-network Planning", *Proc. ICCAD*, pp. 650-665, 2007.

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

[99] C. Li and C.-K. Koh, "Recursive Function Smoothing of Half-Perimeter Wirelength for Analytical Placement", *ISQED*, pp. 829-834, 2007.

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

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

[102] T. Luo, D. Newmark and D. Z. Pan, "A New LP Based Incremental Timing Driven placement for High Performance Designs", *Proc. DAC*, pp. 1115-1120, 2006.

[103] K. Madduri et al., "Parallel Shortest Path Algorithms for Solving Large-Scale Instances", The Shortest Path Problem: Ninth DIMACS Implementation Challenge, pp. 249-290, *DIMACS*, 2009.

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

[105] M. D. Moffitt, J. A. Roy, I. L. Markov, M. E. Pollack, "Constraint-driven Floorplan Repair", *ACM Trans. Design Autom. Electr. Syst.* 13(4), 2008.

[106] G.-J. Nam et al., "A Fast Hierarchical Quadratic Placement Algorithm", *IEEE TCAD* 25(4), 2006, pp.678-691.

[107] G.-J. Nam and J. Cong, "Modern Circuit Placement: Best Practices and Results", *Springer* 2007.

[108] G.-J. Nam, C. C. N. Sze and M. C. Yildiz, "The ISPD Global Routing Benchmark Suite", *Proc. ISPD*, pp. 156-159, 2008.

[109] R. X. T. Nijssen and J. A. G. Jess, "Two-Dimensional Datapath Regularity Extraction", *IFIP Workshop on Logic and Architecture Synthesis* 1996, pp. 110-117.

[110] A. N. Ng et al., "Solving Hard Instances of Floorplacement", *Proc. ISPD*, pp. 170-177, 2006.

[111] M. Pan and C. Chu, "FastRoute: A Step to Integrate Global Routing into Placement", *Proc. ICCAD*, pp. 59-62, 2006.

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

[113] D. A. Papa, S. N. Adya and I. L. Markov, "Constructive Benchmarking for Placement", *ACM GVLSI*, pp. 113-118, 2004.

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

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

[116] S. Reda and A. Chowdhary, "Effective Linear Programming Based Placement Methods", *Proc. ISPD*, pp. 186-191, 2006.

[117] H. Ren, D. Z. Pan and D. Kung, "Sensitivity Guided Net Weighting for Placement Driven Synthesis", *IEEE TCAD*, pp. 711-721, 2005.

[118] A. P. E. Rosiello, F. Ferrandi, D. Pandini and D. Sciuto, "A Hash-based Approach for Functional Regularity Extraction During Logic Synthesis", *Proc. ISVLSI*, pp. 92-97, 2007.

[119] J. A. Roy and I. L. Markov, "ECO-System: Embracing the Change in Placement", *IEEE TCAD* 26(12), pp. 2173-2185, 2007.

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

[121] J. A. Roy et al., "Capo: Robust and Scalable Open-source Min-cut Floorplacer", *Proc. ISPD*, pp. 224-226, 2005.

[122] J. A. Roy, S. N. Adya, D. A. Papa and I. L. Markov, "Min-cut Floorplacement", *IEEE TCAD* 25(7), pp. 1313-1326, 2006.

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

[124] J. A. Roy, J. F. Lu and I. L. Markov, "Seeing the Forest and the Trees: Steiner Wirelength Optimization in Placement", *IEEE TCAD* 23(4), pp. 632-644, 2007.

[125] J. A. Roy et al., "Solving Modern Mixed-size Placement Instances", *Integration* 42(2), pp.262-275, 2009.

[126] A. E. Ruehli, P. K. Wolff and G. Goertzel, “Analytical Power/Timing Optimization Technique for Digital Systems”, *Proc. DAC*, pp. 142-146, 1997.

[127] Y. Saad, “Iterative Methods for Sparse Linear Systems”, *SIAM* 2003.

[128] P. Saxena and B. Hapin, “Modeling Repeaters Explicitly within Analytical Placement”, *Proc. DAC*, pp. 699-704, 2004.

[129] C. Sechen and A. Sangiovanni-Vincentelli, “TimerWolf3.2: A New Standard Cell placement and Global Routing Package”, *DAC*, pp. 432-439, 1986.

[130] 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.

[131] T. Serdar and C. Sechen, “Automatic Datapth Tile Placement and Routing”, *DATE*, pp. 552-559, 2001.

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

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

[134] P. Spindler, U. Schlichtmann and F. M. Johannes, “Kraftwerk2 - A Fast Force-Directed Quadratic Placement Approach Using an Accurate Net Model”, *IEEE TCAD* 27(8) 2008, pp. 1398-1411.

[135] W. Swartz and C. Sechen, “Timing Driven Placement for Large Standard Cell Circuits”, *Proc. DAC*, pp. 211-215, 1995.

[136] T. Taghavi et al., “Dragon2005: Large-scale Mixed-size Placement Tool”, *Proc. ISPD*, pp. 245-247, 2005.

[137] T. Taghavi et al., “Dragon2006: Blockage-aware Congestion-controlling Mixed-size Placer”, *Proc. ISPD*, pp. 209-211, 2006.

[138] L. N. Trefethen and D. Bau “Numerical Linear Algebra”, *SIAM* 1997, pp. 296-298.

[139] N. Viswanathan, M. Pan and C. Chu, “FastPlace2.0: An Efficient Analytical Placer for Fixed-mode Designs”, *Proc. ASP-DAC*, pp. 195-200, 2006.

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

[141] N. Viswanathan et al., “RQL: Global Placement via Relaxed Quadratic Spreading and Linearization”, *Proc. DAC*, pp. 453-458, 2007.

[142] N. Viswanathan et al., "ITOP: Integrating Timing Optimization within Placement", *Proc. ISPD*, pp. 83-90, 2010

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

[144] N. Viswanathan, C. J. Alpert, C. C. N. Sze, Z. Li, Y. Wei, "The DAC 2012 routability-driven placement contest and benchmark suite", *Proc. DAC*, pp. 774-782, 2012.

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

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

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

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

[149] S. I. Ward et al., "Quantifying Academic Placer Performance on Custom Designs", *Proc. ISPD*, pp. 91-98, 2011.

[150] S. I. Ward, D. Z. Pan and E. Swartzlander, "ISPD 2011 Datapath Benchmark Suite", `http://www.cerc.utexas.edu/utda/download/DP/`

[151] S. I. Ward, M.-C. Kim, N. Viswanathan, Z. Li, C. J. Alpert, E. E. Swartzlander Jr. and D. Z. Pan, "Keep it Straight: Teaching Placement how to Better Handle Designs with Datapaths", *Proc. ISPD* 2012.

[152] S. I. Ward, D. Ding and D. Z. Pan, "A High Performance Placer with Automatic Datapath Extraction and Evaluation through High Dimensional Data Learning," *DAC* pp. 756-761, 2012.

[153] Y. Wei et al, "GLARE: Global and Local Wiring Aware Routability Evaluation," *DAC*, pp.768-773, 2012.

[154] E. Wein and J. Benkoski, "Hard Macros Will Revolutionize SoC Design", *EE Times Online*, August 20, 2004. `http://www.eetimes.com/news/design/showArtical.jhtml?articalID=26807055`

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

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

[157] S. J. Wright, "Primal-Dual Interior-Point Methods", *SIAM* 1987, 309 pages.

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

[159] Y. Xu, Y. Zhang and C. Chu, "FastRoute 4.0: Global Router with Efficient Via Minimization", *Proc. ASP-DAC*, pp. 576-581, 2009.

[160] J. Z. Yan et al., "Handling Complexities in Modern Large-scale Mixed-size Circuit Designs", *Proc. DAC*, pp. 436-441, 2009.

[161] C Yang et al., "Physical Synthesis for ASIC Datapath Circuits", *Proc. ICASIC* 2003, pp. 97-100, 2003.

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

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

[164] T. T. Ye et al., "Physical Synthesis for ASIC Datapath Circuits", *Proc. ISCAS*, pp. 365-368, 2002.

[165] Y. Zhang and C. Chu, "CROP: Fast and Effective Congestion Refinement of Placement", *Proc. ICCAD*, pp. 344-350, 2009.

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