On the Use of Directed Moves for Placement in VLSI CAD

by

Kristofer Vorwerk

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2009

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Search-based placement methods have long been used for placing integrated circuits targeting the field programmable gate array (FPGA) and standard cell design styles. Such methods offer the potential for high-quality solutions but often come at the cost of long run-times compared to alternative methods.

This dissertation examines strategies for enhancing local search heuristics—and in particular, simulated annealing—through the application of *directed moves*. These moves help to guide a search-based optimizer by focusing efforts on states which are most likely to yield productive improvement, effectively pruning the size of the search space.

The engineering theory and implementation details of directed moves are discussed in the context of both field programmable gate array and standard cell designs. This work explores the ways in which such moves can be used to improve the quality of FPGA placements, improve the robustness of floorplan repair and legalization methods for mixed-size standard cell designs, and enhance the quality of detailed placement for standard cell circuits. The analysis presented herein confirms the validity and efficacy of directed moves, and supports the use of such heuristics within various optimization frameworks.

Acknowledgements

My heartfelt thanks extend to my wife and family for their patience and encouragement, and to my supervisor and friend, Dr. Andrew Kennings.

To Katrina.

Dans les grandes choses, les hommes se montrent comme il leur convient de se montrer; dans les petites, ils se montrent comme ils sont.

Sébastien Roch

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Overview

As modern integrated circuits (ICs) have grown in size, performance has become limited by the delay of the interconnect rather than the switching speed of logic elements. Computer-aided design (CAD) tools have played an increasingly important role in the development of ICs and in the maximization of design performance of Very Large Scale Integration (VLSI) devices.

CAD tools typically use a sequence of steps (or a "flow") which ultimately transforms a high-level representation of a circuit into a final, routed specification. The algorithms employed in a typical design automation flow can be quite complicated—the optimization problems that must be solved are often NP-complete [50, 78, 118], meaning that the optimal solutions to many of these problems cannot be found in polynomial time but must be solved, instead, via heuristic methods which approximate the optimal solutions.

1.2 Design Styles for Modern VLSI CAD

Computer-aided design tools make it possible to automate the entire VLSI layout process through the use of restricted models and *design styles* which reduce the complexity of the circuit layout. Three design styles which are typically used in modern CAD flows include *standard cells*, *field programmable gate arrays* (FPGAs), and *structured application specific integrated circuits* (ASICs).

1.2.1 Standard Cell Circuits

A standard cell is a logic module with a pre-defined internal layout. These cells have a fixed height but differing widths, depending on the functionality of the module [103]. Standard cells are placed in horizontal rows, with *channels* (or spaces) between rows reserved for interconnect routing. Historically, routing was performed entirely in channels, though in modern circuits, with more layers of metal available for routing, channels are not typically required. Logic modules connect to fixed pads (terminals) which are often placed along the edges of the chip. *Macrocells* are logic modules not in the standard cell format—they are usually larger than standard cells, and may be placed at any convenient location on the chip. In this thesis, circuits with both standard and macrocells are referred to as *mixed-size* designs. Figure 1.1 shows an example of a modern mixed-size circuit layout.

1.2.2 Field Programmable Gate Arrays

In field programmable gate arrays (FPGAs), the entire wafer is prefabricated with a regular grid structure of configurable elements, as shown in Figure 1.2. The placeable elements consist of logic blocks, input/output (I/O) blocks, and intellectual property (IP) blocks, while other configurable elements may exist for programming the routing fabric and configurations of the logic cells.

Logic blocks permit the implementation of a designer's logic. Logic blocks may possess look-up tables (LUTs) and flip-flops (FFs) which allow a designer to implement combinatorial and synchronous logic. In addition to simple logic blocks, modern FPGAs may also implement random access memories (RAMs), carry-chains, embedded processors, or analog circuitry. I/O blocks provide the interface between the internal circuit and the package's pins. A modern FPGA can implement a wide variety of high-speed I/O interface standards.

The interconnect allows routing paths to be configured between individual logic blocks and I/O blocks [19, 48]. FPGAs are usually customized by loading configuration data into internal memory cells. Stored values in these cells determine the logic functions and interconnections in the FPGA. Since clocks are generally distributed to every logic cell in an FPGA, they are routed along dedicated, high-speed lines.

Academic studies in FPGA CAD have historically used a simplified representation of FPGA architectures based on the format popularized by `VPR` [19]. A diagram showing the `VPR`-style architecture and routing resources is shown in Figure 1.3. In this style, I/Os are located around the periphery and core logic is arranged in a sea-of-gates-like fashion, not unlike some commercial architectures (e.g., [1]). The FPGA architecture shown in this diagram possesses uniformly-sized channels, with each I/O slot able to accommodate up to two I/O cells. Each logic cell supports four inputs, corresponding to a cell with a four-input look-up table and a flip-flop.

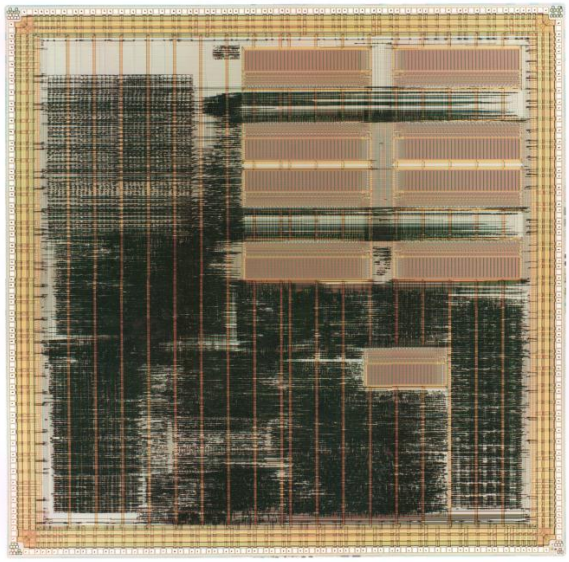


Figure 1.1: Photomicrograph of a modern, mixed-size standard cell layout from [115].

Modern FPGAs are displacing ASICs in many applications due to their ease of use, faster time-to-market, and lower non-recurring engineering costs. Unlike ASICs, FPGAs employ pre-designed routing fabrics that are specifically architected to possess good routability. Conversely, due to their pre-fabrication, FPGAs are not yet capable of achieving the high clock frequencies offered by ASICs.

1.2.3 Structured ASICs

Structured ASICs are a comparatively new VLSI design style. Although the term is somewhat ambiguous, it is generally agreed that structured ASICs "bridge the gap" between FPGAs and full-custom ASICs.

In structured ASICs, the logic mask-layers of a device are predefined by the vendor. Designs are specified through custom metal layers that create connections between lower-layer logic elements (which are chosen from a standard library). A structured ASIC is similar to cell-based ASICs in many ways. For one, the design can be implemented using high-density logic, IP cores, and memory blocks located within the chip fabric. While all layers of a chip are generally customized for each design in cell-based ASICs, the layers of structured ASICs are mostly pre-determined. For example, structured ASICs typically predefine the power, clock, and test structures, as well as base layers of logic, RAMs, and I/Os. These fixed layers simplify many of the issues (such as signal integrity and clock skew) that could otherwise delay the production of a cell-based ASIC.

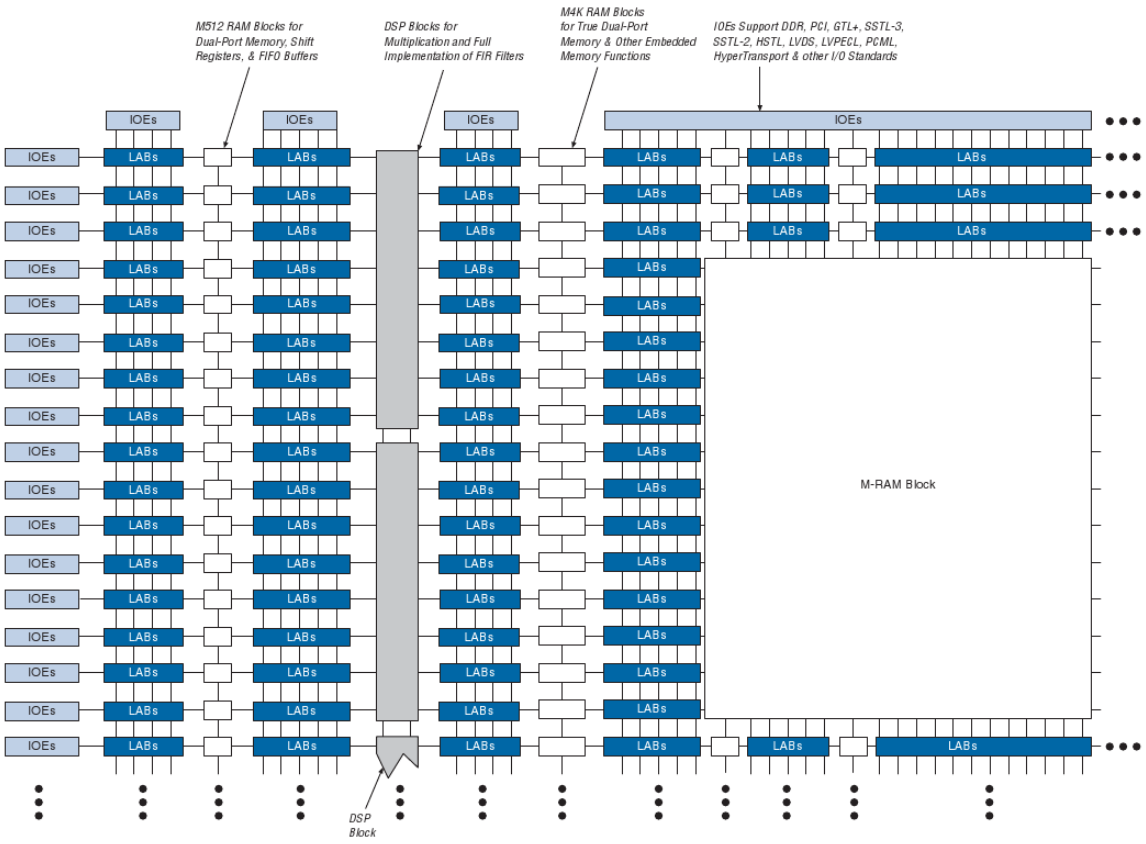
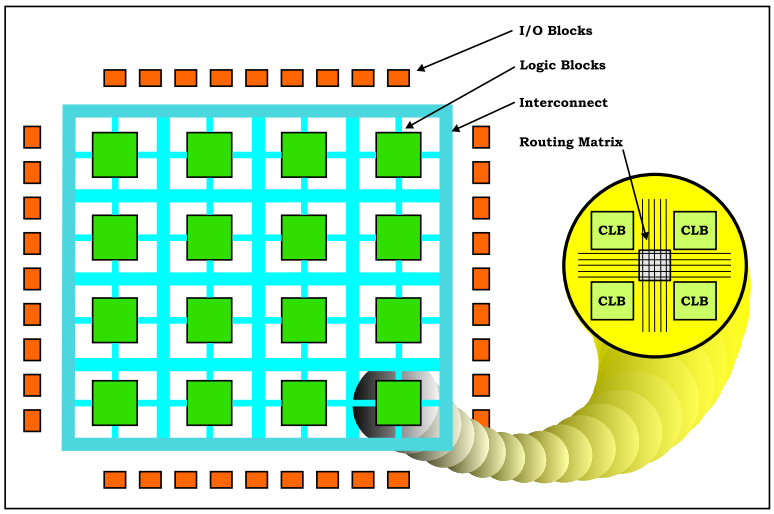


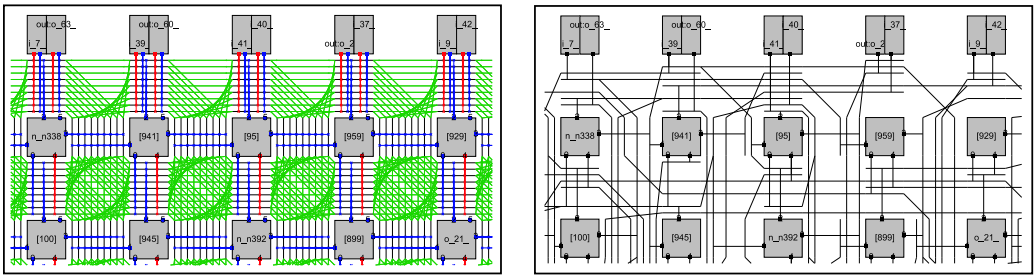
Figure 1.2: Diagram of a modern Stratix FPGA device from Altera Corporation [12], illustrating the presence of RAMs, DSP blocks, I/Os, and core logic cells in the fabric.

Unlike FPGAs, the interconnect in structured ASICs is directly routed, ensuring high performance and low power consumption. Parts of the chip that are not used need not be connected (and, therefore, remain powered down). This allows users to achieve similar densities, speed, and power consumption as a full-custom ASIC design, with lower overall development costs and a shortened development cycle [137]. Structured ASICs are suitable for medium- to high-volume applications which may require higher densities, lower power, or greater performance than can be achieved through FPGAs [133]. A qualitative comparison of the three design styles is shown in Table 1.1.

Altera's HardCopy II devices are low-cost structured ASICs whose pin-outs, densities, and architecture complement Altera's Stratix FPGAs. With the HardCopy service, users can develop and simulate prototype designs on a reprogrammable Stratix FPGA, and later migrate to the "hard-wired" structured ASIC for volume production [41]. An illustration of Altera's HC230 structured ASIC is shown in Figure 1.4. HardCopy II devices are built using an array of fine-grained blocks (called HCells), within a modern process technology. Moreover, they are customized using two metal layers; therefore, configuration circuitry is not required.



(a) FPGA architecture.



(b) Routing resources and sample programmed interconnect.

Figure 1.3: Simplified VPR-style FPGA architecture and interconnect.

HardCopy II devices are nearly equivalent to their FPGA counterparts, but offer significant advantages in terms of power and performance. HardCopy II devices consume less than 50% of the power and offer up to 100% performance improvement over the equivalent Stratix II FPGA due to more efficient use of logic blocks, metal interconnect optimization, die size reduction, and signal buffering [41].

1.3 VLSI CAD Flow

The VLSI CAD flow ultimately aims to implement a user's logic specification in the chosen design style. There are several steps which are reasonably common to all of the aforementioned styles.

The CAD flow begins with a formal specification of a chip. A circuit may be specified, for example, using a schematic or hardware description language such as VHDL or Verilog. The conversion of this high-level representation into a usable, hardware-based implementation

Table 1.1: Qualitative comparison of standard cell, FPGA, and structured ASIC design styles. Structured ASICs tend to combine the best of both design styles, including fast development time and good overall circuit performance.

FPGA	Standard Cell	Structured ASIC
Low NRE Costs	High NRE Costs	Medium NRE Costs
Small-to-Medium Design Size	Large Design Size	Medium Design Size
Easy to Design	Difficult to Design	Easy to Design
Short Development Time	Long Development Time	Short Development Time
Performance Limited	High Performance	High Performance
High Power Consumption	Low Power Consumption	Low Power Consumption
High Per-Unit Cost	Low Per-Unit Cost (at high volume)	Low Per-Unit Cost (at high volume)

occurs during *synthesis*, which converts the specification into a placeable netlist consisting of interconnected technology-specific cells [42, 96]. In deriving this netlist representation, an attempt is made to minimize numerous objectives, including critical path delay, circuit depth, gate-level area, power, and so on.

After synthesis, the modules are *placed* in the IC. Placement seeks to position the netlist cells in valid locations (without overlap) while optimizing criteria such as chip area, wire length, and circuit frequency. The quadratic assignment problem (QAP) can be viewed as a simplified special case of the placement problem, with a worst-case complexity of $O(n!)$ [50] in the number of modules. In practice, however, placement is substantially more complex due to the presence of overlap constraints, the freedom with which cells may be placed in a die, and the presence of cost functions which cannot be computed in polynomial time and must therefore be approximated using heuristic approaches. The quality of these heuristics largely determine the performance and area requirements of the final, integrated circuit.

After the cells have been placed, the circuit is *routed*. This process establishes the pin-to-pin connections between cells. Finding an optimal routing given a placement is also a NP-complete problem [103], although a number of heuristic approaches exist (cf. [19, 84]) which can find very good, admissible routes in polynomial time.

The final steps in the VLSI CAD flow typically involve verification of the circuit layout. These steps may consist of a "design rule check" to ensure that layout is legal and a layout-versus-input check to ensure that the implemented design satisfies the original functionality.

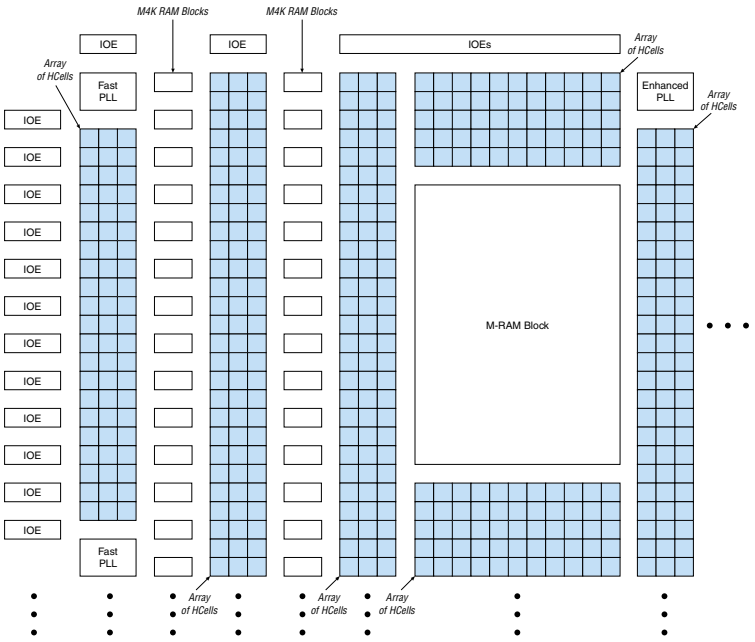


Figure 1.4: Partial floorplan of an HC230 structured ASIC from Altera Corporation [41]. Heterogeneous resources including I/Os, M4K RAMs, PLLs, and Mega-RAMs must be placed into disjoint slots.

1.4 Motivation and Contributions

Across each of the design styles, placement remains one of the most influential steps in the CAD flow—it is directly responsible for determining the relative locations of modules and (indirectly) for establishing the lengths of the routes between them. The decisions made during placement substantially determine design performance, power, routability, and area. For example, in a study of FPGA placement and routing heuristics [88], a finely-tuned placer was found to yield critical path delays which were up to three times smaller, on average, than those produced by a naive, random scattering strategy. Given its influence over solution quality, placement is, therefore, a viable avenue in which to investigate ways for improving vital design characteristics, and is the focal point of this thesis.

Despite recent advances in the literature, there remains substantial room for improving placement heuristics [28]. This work examines strategies for improving placement by enhancing stochastic search methods—such as simulated annealing—via the concept of *directed moves*. These moves help to guide search-based optimization strategies by focusing efforts on moves which are most likely to yield productive improvement, effectively pruning the size of the search space. This thesis presents the engineering theory and implementation of directed moves, and

documents ways in which they can be used to improve the quality of global placements for FPGAs, improve the quality of floorplan repair and legalization methods for mixed-size ASICs, and improve the quality of detailed placements for standard cell ASICs.

1.4.1 Directed Moves for FPGAs

Simulated annealing remains a widely-used heuristic for FPGA placement due, in part, to the flexibility with which the annealing objective can be adapted to handle realistic architectural constraints. For example, modern FPGA CAD software typically supports user-definable constraints (such as "LogicLock" regions [41]) which are easily modelled by an annealing-based placer.

However, as FPGAs continue to grow in size, the large run-times incurred by simulated annealing are becoming prohibitive. To limit the search space, practical implementations perform random perturbations of logic within range-limited windows [19]. Such "simple moves" are inexpensive, but many such moves must be performed to achieve good quality. In this work, several directed moves are described which help to guide an FPGA placer to consistently produce better-quality solutions for the same amount of run-time as previous techniques. Moreover, a technique for automatically computing the most "effective" move is described—it is also shown how such a mechanism can be incorporated into the placement heuristic, as well as how the technique can be used as a termination criterion for the anneal.

1.4.2 Directed Moves for Mixed-Size Legalization

Traditionally, mixed-size placement methods have employed a *two-stage* approach, wherein a heuristic method is employed to produce an initial (or "global") placement, followed by overlap removal. In such methods, detailed cell overlaps are ignored in the global placement and later resolved by a "legalizer" once a sufficient cell distribution has been achieved [2, 8, 27, 27, 34, 60, 64, 128]. Legalizers attempt to preserve the global placement as much as possible, as doing so preserves the objectives sought by the global placer, such as whitespace and density constraints for routability. Owing to larger circuit sizes and oddly-shaped blocks, modern circuits render many previous legalization techniques [8, 20] unreliable for producing legal solutions [86, 94], or potentially destructive in the sense that tightly-packed cells may not be desirable for routability (cf. [91]).

These observations motivate this work on *legalization* which addresses the second phase—overlap removal—of the floorplacement problem. This thesis demonstrates that a straightforward, top-down approach for legalizing circuits, when combined with a local search strategy employing directed moves, can *reliably* produce feasible placements and floorplans with excellent quality and

run-times compared to leading academic tools. The method introduced in this work perturbs only those features which are responsible for violating overlap constraints.

1.4.3 Directed Moves for Detailed Placement

Continuous placement methods have been studied as a means of placing cells for almost forty years. These methods offer several alluring benefits, such as excellent run-time scalability, good quality, and amenability toward engineering change-order (ECO) optimization. However, these methods suffer from the drawback that they cannot precisely model complicated objective functions as part of their optimization strategy. Yet, the literature in mixed-size placement is mostly bereft of discussion on optimizing combinations of objectives at the same time, such as both wire length *and* overlap.

Instead, many complicated optimizations—such as whitespace insertion—are performed after legalization in a step known as *detailed placement*. Modern CAD theory has espoused the use of branch-and-bound-based strategies for optimizing cells within very localized windows during detailed placement. Simulated annealing and other search-based placement methods have been all but abandoned because of the belief that they offer poor scalability for modern standard-cell problems.

Despite this commonly-held belief, simulated annealing is shown, in this work, to be an effective strategy for detailed placement. The improvements described herein are borne from the use of directed moves within the annealer, and a strategy for maintaining legality via cell rippling during placement. This work shows that directed moves can improve the quality of placements produced by a standard cell annealer without harming run-time.

1.5 Organization

The rest of this thesis is organized as follows. Chapter 3 introduces the concept and theory of directed moves, and describes the implementation of several such moves within an academic FPGA annealing framework. Chapter 4 describes the mixed-size legalization problem, and presents a novel solution employing a top-down optimization strategy coupled with directed moves. Chapter 5 examines the use of simulated annealing with directed moves for detailed standard cell placement. Finally, Chapter 6 summarizes the work, and offers concluding remarks.

Supplementary information is provided in the form of appendices at the end of the thesis. Appendix A presents a list of the papers that were published as a result of the work described herein. Appendix B motivates the use of multiple seeds when reporting the results from annealing-based placement. Finally, Appendix C presents an overview of the data structures related to this work.

CHAPTER 2

Background

The quality of a global placement can effect a tremendous change in the overall performance of an integrated circuit. Recent experiments suggest that placement tools yield results that are 50%–150% worse than optimal [38]. On the other hand, the demand for higher-quality placement techniques must also be balanced with the need for shorter run-times.

This chapter begins, in Section 2.1, with a brief review of the most common strategies for placement, including search-based, partitioning-based, and analytic methods. Subsequently, Section 2.2 presents specific details pertinent to the different design styles examined in this thesis, with emphasis on FPGA placement, mixed-size legalization, and standard cell detailed placement.

2.1 General Overview of Placement

Placement is a critical step in VLSI physical design and the focal point of discussion in this thesis. As problem instances have increased in size and complexity, placement has, more and more, become the bottleneck in deep sub-micron designs. Typically, placement seeks to minimize wire length and critical path delays subject to the constraints that cells must be placed into prescribed locations without overlap.

Placement typically begins with a circuit netlist modelled as a hypergraph $G_h(V_h, E_h)$ with vertices $V_h = \{v_1, v_2, \ldots, v_n, v_{n+1}, \ldots, v_{n+p}\}$ representing circuit cells and hyperedges $E_h = \{e_1, e_2, \ldots, e_m\}$ representing circuit nets. The set $\{v_1, v_2, \ldots, v_n, \}$ represents movable cells and the set $\{v_{n+1}, \ldots, v_{n+p}\}$ represents pre-placed cells and I/O pads. Each vertex v_i has dimensions w_i and h_i that represent the width and height of its corresponding circuit cell, respectively. Let (x_i, y_i) denote the coordinates of the centre of vertex v_i. Placement information is then captured in the x- and y-directions by two vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (x_1, x_2, \ldots, x_n)$.

2.1.1 A Brief Overview of Placement Objectives

Depending on the design style and goal, a placement heuristic may optimize different objectives.

2.1.1.1 Half-Perimeter Wire Length

Wire length is one of the most commonly-employed measures of quality in standard cell and FPGA placement—the minimization of wire length can lead to improvements across multiple objectives (such as routability and circuit performance). However, the precise wire length of a net can be impractical to compute for large-scale placement; instead, an approximation that is closely correlated to the interconnection length is required.

For this purpose, the approximation most commonly employed in modern placement is that of the *half perimeter—or "bounding box"—wire length* (HPWL) which, for any given net $e \in E_h$, is half of the perimeter of the minimum rectangle that encloses all cells on net e. The HPWL of a net e can be written as

$$\text{HPWL}(e) = (\max_{j \in e} \mathbf{x}_j - \min_{j \in e} \mathbf{x}_j) + (\max_{j \in e} \mathbf{y}_j - \min_{j \in e} \mathbf{y}_j) \tag{2.1}$$

where $\mathbf{x}_j$ represents the location of module j (connected, in this case, to net e) in the x-direction, and similarly for the y-direction. The total half perimeter wire length of the circuit is given by $\sum_{e \in E_h} \text{HPWL}(e)$.

The complexity of evaluating and minimizing HPWL depends largely upon the circuit's structure and the chosen placement technique. For a circuit in which every terminal is connected to every net in G_h, a straightforward evaluation of (2.1) has a worst-case complexity of $O(|V_h||E_h|)$. Of course, this worst-case complexity is highly-dependent upon the structure of the netlist—in sparsely-connected circuits, the complexity of evaluating (2.1) may approach $O(|E_h|)$.

The chosen placement methodology also plays a contributory role in the complexity of evaluating and optimizing HPWL. In simulated annealing-based placement (see Section 2.1.2), Equation (2.1) need only be computed once in entirety at the beginning of placement. (This computation can be made with the aforementioned worst-case complexity, since module locations are generally known during annealing.) Thereafter, the HPWL can be updated incrementally for only those nets which are perturbed during the placement; this incremental computation is accelerated, in practice, by maintaining a cache of the bounding boxes of the nets. Since only a small fraction of the nets in a design are modified at a given time by the annealer, the incremental evaluation of HPWL can be performed quickly. On the other hand, a large number of these perturbations may be required in order to achieve a high-quality placement result.

In partitioning-based techniques (see Section 2.1.3), the precise locations of all modules may

not be known during placement. As a result, it can be difficult to precisely evaluate (2.1). To compensate for this imprecision, partitioning-based strategies typically employ an *approximation* to HPWL based on a minimum-cut heuristic [23]. Efficient minimum-cut hypergraph bi-partitioning heuristics, such as Fiduccia-Mattheyses [49], have linear-time complexity [103]; however, the need for recursive bi-partitioning (to spread cells across the placement) and for multiple applications of the partitioning heuristic (to ensure high-quality cut solutions) add to the complexity of the technique.

Analytic and force-directed approaches (see Section 2.1.4) employ mathematical formulations to minimize HPWL during placement. Historically, the mathematical minimization of HPWL was accomplished using network flow or linear programming; however, these methods suffer from poor run-time scalability. Modern approaches to mathematical placement convert the circuit hypergraph to a weighted graph, whereupon the placement technique can employ efficient Newton-type methods with a quadratic (or linearized quadratic) wire length objective. While quickly solvable, such objectives may be poorly correlated with the HPWL of the circuit. The work of [70] presented an analytical method for HPWL minimization that did not rely on a hypergraph-to-graph conversion—instead, a family of smooth and everywhere-differentiable functions were presented which were shown to approximate HPWL arbitrarily closely. The contributions of [70] laid the foundation for a number of techniques based on differentiable approximations to HPWL, some of which are reviewed in Section 2.1.4. In general, the complexity of these methods lies primarily in the way in which cell overlap is minimized during placement rather than the actual evaluation of the function used to approximate (2.1).

2.1.1.2 Critical Path Delay

Critical path delay minimization is another common placement objective, particularly in FPGA CAD. Circuit timing is generally performed on a timing graph [19] which models the interconnections and delays in the circuit. Timing analysis is computed using a method akin to CPM analysis [132]. To perform timing analysis, a directed graph $G(V,E)$ is constructed to model the delays in the circuit. Pins on logic blocks become nodes in the graph. Nets in the netlist become directed edges between nodes. Every edge is annotated with a physical delay. In the simplest context, a *primary output* is the pin of an output pad, while a *primary input* is the pin of an input pad. Register inputs and outputs can be considered as pseudo-primary outputs and inputs, respectively.

A simplistic timing computation can be explained as follows. Given a node j, its *arrival time*,

$\text{Arr}(j)$ can be computed from the equation:

$$\text{Arr}(j) = \begin{cases} 0, & j \in \text{primary outputs} \\ \max\{\text{Arr}(i) + \text{Delay}(i,j)\}, & (i,j) \in E \end{cases} \tag{2.2}$$

where $\text{Delay}(i,j)$ is the delay of the edge connecting nodes i and j. The maximum arrival time of all nodes, $\text{Delay}_{\max}$ can be computed as:

$$\text{Delay}_{\max} = \max \text{Arr}(j), j \in \text{primary inputs} \tag{2.3}$$

Given a node i, the *required time* of the node can be computed as:

$$\text{Req}(i) = \begin{cases} 0, & i \in \text{primary inputs} \\ \min\{\text{Req}(j) - \text{Delay}(i,j)\}, & (i,j) \in E \end{cases}. \tag{2.4}$$

The *slack* of an edge can then be computed as:

$$\text{Slack}(i,j) = \text{Req}(j) - \text{Arr}(i) - \text{Delay}(i,j). \tag{2.5}$$

The *critical path* is the path with the worst slack, and can be viewed as the path which limits the maximum performance of a design. discussion ignores the case of *false paths*. False paths arise during static timing analysis—they are valid paths in terms of the interconnecting circuit, but are unlikely to transmit signals during normal operation due to the circuit's. They can arise during static timing analysis because a graph-based timing analyzer may not simulate or correctly model the actual switching behaviour of the circuit. Worst-case slack maximization—alternatively referred to as critical path optimization—is, therefore, a common goal in timing-driven techniques.

Timing-driven placement algorithms can generally be viewed as being either net-based or path-based. Path-based algorithms attempt to compute the delays of all paths and minimize the longest path delay directly [77, 138], whereas net-based algorithms transform timing constraints into weights. In the latter approach, timing analysis is performed at specific times throughout the placement, and the weights on nets are adjusted to reflect the updated information.

It is worth noting that timing computation in commercial tools can be substantially more complicated than in the academic literature, as such tools typically account for multiple clock domains, rising and falling edge triggering, false paths, and so forth. In particular, this thesis ignores the effects of false paths which "are paths that should not be considered during timing analysis or which should be assigned low (or no) priority during optimization" [13]. Furthermore, this thesis considers only static timing analysis, which relies on the interpretation of circuit

performance from the perspective of the timing graph, and does not incorporate simulation to identify false paths (as is done during dynamic timing analysis).

2.1.2 Simulated Annealing-Based Placement

Search-based placement methods involve random, iterative improvement of an existing solution. For example, simulated annealing-based placers, such as `TimberWolf` [110, 113] and `VPR` [19], produce placements using stochastic search (cf. [75]). Genetic algorithms are another type of search-based heuristic which work by "emulating the natural process of evolution as a means of progressing toward the optimum" [103]. Genetic algorithms are generally not used in modern CAD flows, and are not treated here.[1]

Simulated annealing is perhaps the most well-developed, well-studied method for module placement. It can be time-consuming, but can yield good results. Most importantly, the *cost function* used in an annealer can easily be extended to consider new constraints (such as overlap removal or thermal "hot-spot" minimization) with only *minimal changes* required to the remainder of the placement flow.

As a result of its flexibility, simulated annealing remains a widely-used heuristic for placement in tightly-constrained design styles, such as FPGAs. FPGAs tend to impose more constraints on the validity of cell locations than in standard cell designs—for instance, the placement of basic logic elements in modern FPGAs is constrained by the pre-fabricated routing resources available for each logic block. Thus, it does not merely suffice, in some FPGAs, to ensure that cells are placed in non-overlapping locations, but also that the wires connecting logic blocks do not exceed IC limitations. An annealing-based placer, more than any other placement technique, can quickly and easily be modified to account for such constraints.

Simulated annealing is essentially an improvement of a random pairwise interchange algorithm. In this approach, the heuristic periodically accepts moves that result in an increase in the cost in an effort to prevent the method from becoming stuck in local minima. The heuristic works as follows. All moves that result in a reduction in the cost are accepted. Moves which result in a cost increase are accepted with a probability that decreases with the increase in cost [103]. A *temperature* parameter T is typically used to control the probability of accepting moves which increase cost. In most implementations, the acceptance probability is given by $e^{\frac{-\Delta C}{T}}$, where ΔC is the increase in cost [103]. Initially, the temperature is set to a large value, allowing numerous cost-increasing moves to be accepted. The temperature is then gradually decreased, so that the probability of accepting a cost-increasing move is also decreased. If left to run for

1 The reader is referred to [103] for more information about genetic placement.

a sufficiently long time with a proper cooling schedule, simulated annealing can converge to the global minimum [85].

Simulated annealing derives its name from the annealing process in metals [75]. If a metal has an imperfect crystal structure, its atomic arrangement can be restored by heating it to a high temperature and then allowing it to cool slowly. At high temperature, the atoms have sufficient kinetic energy to break loose from their incorrect positions. As the material cools, the atoms become trapped at the correct lattice locations. If the material is cooled too rapidly, the atoms may not move into correct lattice locations, thereby freezing defects into the crystal structure [103]. Analogously, in annealing-based placement, the high initial temperature T allows cells at incorrect initial locations to be dislodged from their positions. As T decreases, the cells are placed into their optimum locations.

The pseudocode for a typical simulated annealing-based placer is shown in Figure 2.1. Initially, the cells in the netlist N are placed in random (but valid) locations. Within the inner loop, modules are either randomly displaced to new locations or interchanged. A range-limiting function may be applied to ensure that cells are not moved further than a specified distance from the target location [103]. The change in cost is computed for a move by evaluating the change in only those nets connected to cells that were moved. If the cost improved after the perturbation, the new cell locations are retained. Otherwise, if the cost worsened, the new placement may still be retained (probabilistically) based on the current temperature, T. The temperature for the next loop of the algorithm is subsequently decreased based on the number of iterations and the previous temperature. The temperature at iteration $i+1$, for example, may be derived simply by taking a fraction of the temperature in iteration i, as in $T_{i+1} = \alpha \cdot T_i$, for $0 < \alpha < 1$. In simulated annealing, there "are no fixed rules about the initial temperature, the cooling schedule, the probabilistic acceptance function, or the stopping criterion, nor are there any restrictions on the types of moves to be used—displacement, interchange, rotation, and so on" [103].

2.1.3 Partitioning-Based Placement

A top-down, divide-and-conquer approach to global placement has been used successfully in commercial tools for many years. This approach "seeks to decompose the given placement problem instance into smaller instances by subdividing the placement region, assigning modules to sub-regions, reformulating constraints, and cutting the netlist—such that good solutions to smaller instances (sub-problems) combine into good solutions of the original problem" [23]. That is, top-down methods recursively divide the placement area and the circuit netlist into smaller pieces using either bi-section or (less commonly) quadri-section and a minimum-cut (or other) objective function to approximate wire length.

```
   Procedure: SIMULATED ANNEALING
   Input: A netlist, N
1  begin
2    Initialize variables;
3    Generate a random placement of the cells in N;
4    while outer_loop_count < MAX_OUTER_PASSES and exit criteria not satisfied do
5      inner_loop_count ← 0;
6      while inner_loop_count < MAX_INNER_PASSES do
7        Perform a random perturbation of the placement;
8        ΔC ← the change in cost of the placement;
9        if ΔC < 0 or the probability function accepts the move then
10         Accept the new placement;
11       else
12         Reject the new placement (and restore the previous cell location);
13       fi
14       T ← decreased value based on cooling schedule;
15       inner_loop_count ← inner_loop_count + 1;
16     od
17     outer_loop_count ← outer_loop_count + 1;
18   od
19 end
```

Figure 2.1: Pseudocode for a general simulated annealing placement algorithm.

Modern partitioning-based placers decompose the netlist using minimum-cut hypergraph bi-partitioning. Quadri-section techniques [109] are less commonly used in modern flows. Each bi-partitioned instance is created from a division of a rectangular region, or *block*, in the placement region. Figure 2.2 shows an example of a placement region partitioned alternately using horizontal and vertical cuts. At each level, the number of nets intersected by the cut line is minimized, and the sub-circuits are assigned to horizontally and vertically partitioned chip areas [103]. Pseudocode for a general top-down, partitioning-based placement heuristic is provided in Figure 2.3. This pseudocode provides a high-level outline of the placement strategy.

Inside each block, there exist nodes which correspond to the cells inside the block as well as propagated external terminals. These terminals represent the connections from cells internal to the block to modules external to the block. Such modules may exist in another partitioned region, for instance. The modules are *propagated* to a block's boundaries to account for the external connections. The motivation for doing so follows from the notion that, if a module is "connected to an external terminal on the right side of the chip, it should be preferentially assigned to the right side of the chip, and vice-versa" [103]. To propagate terminals, the partitioning must be done in a breadth first manner—there is little point in partitioning one group to finer levels without

partitioning the other groups, since in that case, no information would be available about the group to which a module should preferentially be assigned [103].

Cell placement imposes additional constraints on the partitioning of a hypergraph—chiefly, that the sizes of the partitions in the solution are not allowed to deviate from target partition sizes. These constraints arise because the proportion of whitespace in modern designs is often quite small. Thus, the total module area assigned to a block must closely match the available layout area in the block [23]; otherwise, relaxed balance constraints can lead to uneven area utilization and overlapping placements [10, 23].

Partitioning is typically performed using an iterative, multi-level Fiduccia-Mattheyses (FM) heuristic [22, 49]. The Kernighan-Lin (KL) heuristic [45, 72] is also used for hypergraph bi-partitioning. For example, the popular multi-level partitioner, `hMetis` [67, 68], employs FM for large partitioning instances, while KL is used when the instances are smaller than a threshold parameter.

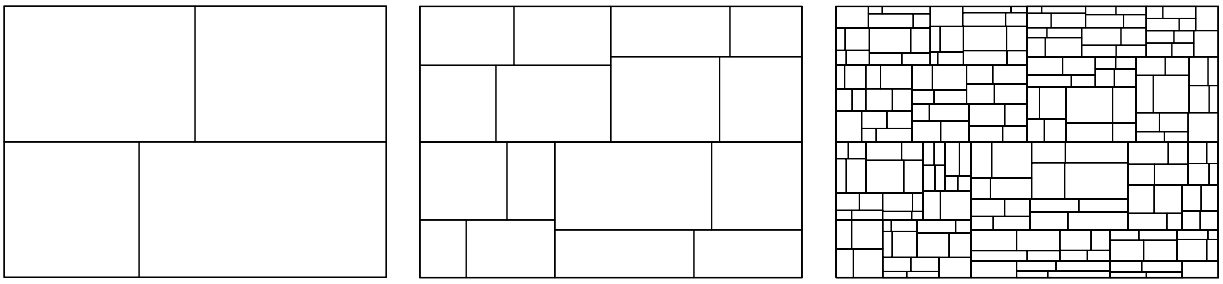
Once a partitioned block is sufficiently small (or contains too few cells), partitioning-based placers use an alternate, *end-case* algorithm to finalize cell locations. Tight balance constraints and a potentially large variation in standard cell sizes makes small partitioning instances difficult for a FM partitioner to solve. This problem arises in small instances because the FM algorithm "may (1) never reach the feasible part of the solution space (especially if it has trouble finding an initial balance-feasible solution) and (2) even a relative scarcity of feasible moves (from any given feasible solution) can make the algorithm more susceptible to being trapped in a bad local minimum" [23]. Consequently, a branch-and-bound strategy is typically employed for small partitioning instances (cf. [21, 23]).

With the advent of multi-level hypergraph partitioning in [68], the quality of cuts generated by partitioners improved significantly, and by extension, so did the quality of VLSI placements.[2] Since then, hundreds of papers have undertaken the task of improving upon partitioning-based techniques; the following is a selection of some of the most pertinent works.

In [23], the authors examined end-case partitioning strategies, as well as a branch-and-bound technique for optimal cell placement. The authors point out that FM-like strategies do not work well for end-case placement (when block sizes are too small) due, in part, to tight area-balancing constraints. Instead, enumerative approaches can yield significantly better cuts for small blocks. Since then, almost all partitioning-based placers have employed similar strategies for end-case placement.

In [130], Vygen considers a method of quad-secting a placement region using American maps and a linear-time binary-search-like heuristic. The author describes how a region of already-placed

2 An excellent review of partitioning and its applications to placement prior to 1995 is given in [10].



(a) One level (b) Two levels (c) Up to four levels

Figure 2.2: Placement region partitioned using alternating horizontal and vertical cuts. In this diagram, a "level" refers to one horizontal followed by one vertical cut of each partitioned block. Generally, iterative partitioning of a block stops when the size of the block or the number of cells contained therein passes a threshold parameter. For such blocks, a different end-case partitioning strategy is often employed.

cells (with a given weight, and a given capacity per region) can be quad-sected such that the total weight of points assigned to a quadrant does not exceed its capacity and the total movement is minimized. Vygen proves that, at most, only three cells may be "split" and partially assigned to several quadrants [130]. This technique forms the basis for the `BonnPlace` [129] placement tool.

Caldwell et al. introduce a recursive bisection placement tool in [24]. This paper builds upon the authors' previous work on multi-level hypergraph partitioning and end-case placement (cf. [21–23]), and describes the implementation of the first commercial-quality, academic partitioning-based placer, `Capo`. Since then, `Feng Shui` [7] has emerged as another bisection-based placer. The two differ primarily in their placement of horizontal cuts: while `Capo` attempts to place horizontal cuts along standard cell row boundaries to aid cell legalization, `Feng Shui` allows cuts to occupy a fraction of a row (a "fractional cut"). The latter then employs a row-by-row legalization strategy after global placement to satisfy overlap constraints.[3]

Kahng and Reda, in [63], introduce a concept called "feedback", which proposes a solution to the problem of ambiguous terminal propagation. The concept of augmenting partitioning-based techniques to determine how to propagate terminals (when there remain partitioning blocks which have not yet been processed) is similar in motivation to [5]. In this work, however, the blocks at a given level of the partitioning are placed via minimum-cut bisection, and then repeatedly restored and *replaced* using the previous iteration's cell locations to intelligently propagate terminals. While feedback can slow the partitioning process by causing blocks at each level to be repartitioned multiple times, a significant improvement in overall placement quality can result.

3 The topic of legalization is treated in more detail in Section 2.2.2.

```
   Procedure: TOP-DOWN PARTITIONING PLACEMENT
   Input: A netlist, N
   Local Variables: A queue of blocks
1  begin
2    Initialize a block which contains all cells in N and has the original
3      placement region as its dimensions;
4
5    while queue is not empty do
6      Dequeue a block;
7      if block is small enough then
8        Use end-case placement to place cells in block;
9      else
10       Bi-partition cells from block into two smaller sub-blocks;
11       Enqueue both sub-blocks;
12     fi
13   od
14 end
```

Figure 2.3: High-level pseudocode for a top-down partitioning-based placement technique [23].

2.1.4 Analytic and Force-Directed Placement

Analytic placement methods (cf. [46, 47, 76, 104, 117, 129]) use linear or quadratic optimization to place cells.

Although convex, HPWL is neither a strictly convex nor differentiable function, and is therefore difficult to minimize directly. As a result, analytic methods typically select a different (but satisfactory) approximation for efficient minimization. One of the most popular approximations to HPWL is that of quadratic wire length. While linear programming formulations themselves are generally not employed for global placement, various other techniques have been used to approximate a linearized objective (cf. [9, 14, 69, 70, 79]).

In [54], Hall formulated the placement problem as a quadratic assignment problem (QAP) and devised a method for solving it using eigenvalues. By itself, the quadratic assignment problem is arguably the most difficult NP-hard combinatorial optimization problem—solving general problems of size greater than thirty is still computationally impractical due, in part, to the lack of sharp lower bound techniques [53].

The quadratic assignment problem is formulated as follows: given a cost matrix $\mathbf{C}_{ij}$ representing the connection cost of elements i and j and a distance matrix $\mathbf{D}_{kl}$ representing the distance between locations k and l, find a permutation function p that maps elements i and j to locations

$k = p(i)$ and $l = p(j)$ such that the sum

$$\phi = \sum_{i,j} \mathbf{C}_{ij}\mathbf{D}_{p(i)p(j)} \tag{2.6}$$

is minimized [103]. Hall showed that cell placement could be converted to a quadratic assignment problem, with $\mathbf{C}_{ij}$ representing the connectivity between cell i and cell j, and $\mathbf{D}_{kl}$ representing the distance between slot k and slot l. The permutation function p maps each cell to a slot. The wire length is given by the product of the connectivity and the distance between the slots to which the cells have been mapped [103]. Thus, ϕ gives the total wire length for the circuit, which is to be minimized [103]. Since the cost function seeks to minimize the square of the distance between logic cells, this method is known as *quadratic placement*.

Requiring logic cells to be placed into fixed slots leads to a series of n equations which restrict the values of the logic cell coordinates [35, 107]. If all of these constraints are imposed, the quadratic problem becomes NP-hard. Instead, Hall proposed that these constraints be relaxed. This leads to an approximation of the QAP placement which can be solved very quickly; however, the consequence of this relaxation is that cells may overlap. To overcome this overlap, quadratic methods are often augmented with "spreading forces", as discussed in the following sections.

2.1.4.1 Quadratic Placement

An alternative quadratic formulation was introduced in [76]. In this approach, the overall method for minimizing wire length is accomplished by solving the quadratic optimization problem (x-direction only) given by

$$\min_x \left(\sum_{i,j} a_{ij}(x_i - x_j)^2 \right) = \min_x \frac{1}{2}\mathbf{x}^T\mathbf{Q}_x\mathbf{x} + \mathbf{c}_x^T\mathbf{x} + \mathbf{d}_x \tag{2.7}$$

where a_{ij} represents the weight of the edge connecting cells i and j in the weighted graph representation of the circuit. A similar optimization problem is solved for the y-direction. The matrix $\mathbf{Q}_x$ is the Hessian which encapsulates the hyperedge connectivities. Assuming that some cells are fixed, the Hessian is a symmetric, positive-definite matrix. This requirement is realized in any real circuit since I/O pads are fixed, typically around the periphery of the placement area. The vector $\mathbf{c}_x$ is a result of fixed cell-to-free cell connections, and the vector $\mathbf{d}_x$ is a result of fixed cell-to-fixed cell connections.

This optimization problem is strictly convex and has a unique minimizer given by the solution

of a single, positive-definite system of linear equations (x-direction only),

$$\mathbf{Q}_x\mathbf{x} + \mathbf{c}_x = \mathbf{0}.$$

In this formulation, cell overlap is ignored, and the vector $\mathbf{x}$ provides only relative cell locations. An example of the highly-overlapping nature of a quadratic placement is shown in Figure 2.4.[4]

2.1.4.2 Hybrid Methods

Multiple techniques are often combined to improve the performance and quality of the resulting placements, as well as to handle additional constraints in a convenient manner. For example, `Dragon` [82] uses recursive partitioning to arrive at an initial placement, which is then improved using simulated annealing methods.

Similarly, `GORDIAN` [76], `GORDIAN-L` [104] and `BonnPlace` [129] combine quadratic formulations with top-down partitioning-based methods. In such frameworks, analytic techniques are used to solve a relaxed placement problem to determine relative cell locations while ignoring placement restrictions—that is, cells are allowed to overlap. Partitioning-based methods are subsequently employed to enforce the constraints that cells must not overlap with each other while further optimizing the placement.

In [11], the authors describe a means of augmenting partitioning-based placement using analytic strategies. In their work, a quadratic placement is used to calculate the area balance parameter for dividing each block—this parameter is then used to make a more informed minimum-cut partition. The significance of this paper is that it presents a means of enhancing the quality of cuts using analytic methods.

Adya et al. further the notion of analytically-augmented partitioning-based placement in [5]. In this work, the authors use quadratic placement to aid in propagating terminal cells within each block. First moment constraints (based on the area of cells within each block) are added to the quadratic problem to encourage cell spreading (cf. [76]). The locations of the cells from the quadratic placement are then used to aid in the assignment of propagated terminals for partitioning blocks which have not yet been processed.

2.1.4.3 Force-Directed Methods

Alternatively, an analytic method can use *forces* such that fairly non-overlapping placements are obtained without the need for partitioning. Force-directed methods have been studied over the past four decades as a means of placing cells. The common denominator in these methods is

4 The reader is referred to [54, 71, 76, 117, 121] for more information about the quadratic problem formulation.

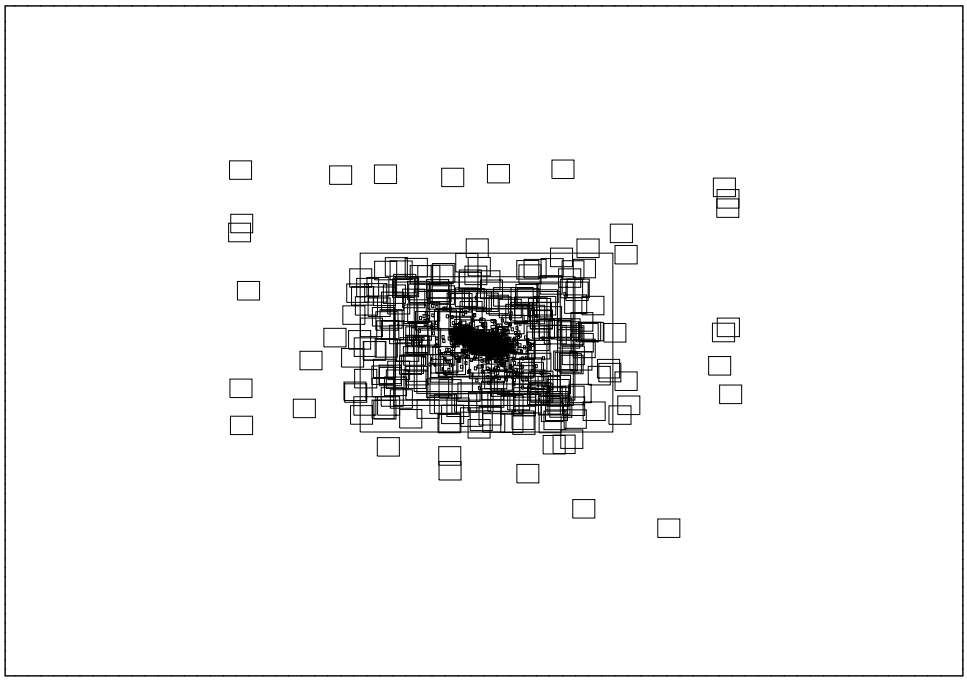


Figure 2.4: Quadratic placement for a mixed-size problem with approximately twenty-seven thousand cells. I/O pads, fixed along the periphery, "pull" some of the cells outward from the centre, but the placement is still far from legal.

that "forces" are used to calculate the cells' positions to achieve an objective such as shorter wire length or smaller delay. The use of forces is borne out of the physical analogy with Hooke's law for stretched springs, wherein connected cells can be viewed as exerting attractive spring forces on one another. The magnitude of the force between any two cells is directly proportional to the distance between them. If the cells in such a system could move freely, they would move in the direction of their forces until the system achieved equilibrium at a minimum energy state. Unfortunately, a minimum energy placement is most often not valid as cells have physical dimensions which are ignored in the spring analogy. Consequently, additional forces are applied to perturb the cell positions and remove overlap. Force-directed methods, in general, purge cell overlap over many iterations while trading off attractive and repulsive forces to achieve a placement in which cells are distributed evenly without overlap. For example, the progress made by a force-directed placer on circuit `ibm04` from the ICCAD04 mixed-size placement benchmark suite [2] is illustrated in Figure 2.5.

Force-directed methods differ from simulated annealing and partitioning-based methods. Simulated annealing typically begins with an initial feasible (or nearly feasible) placement and applies iterative improvement. Minimum-cut and partitioning methods are also constructive,

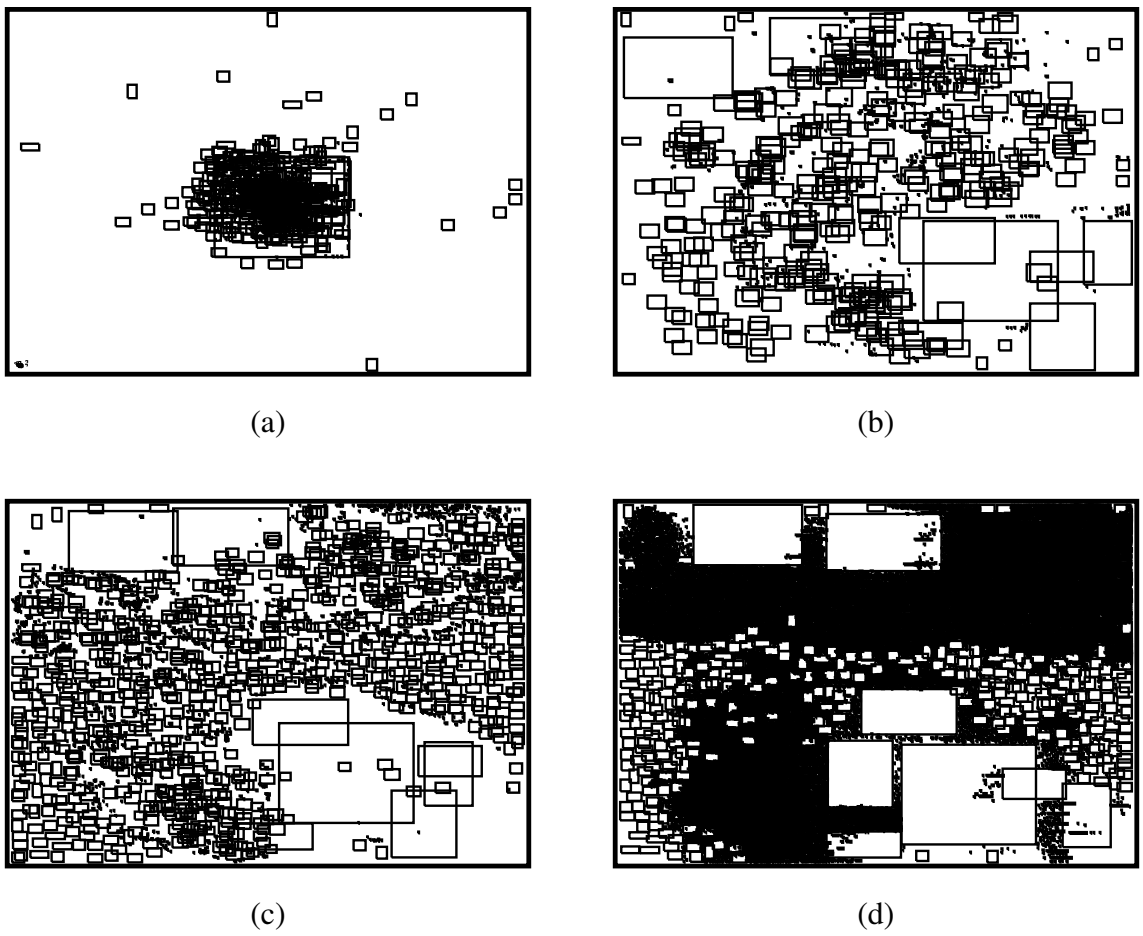


Figure 2.5: Typical progression of a continuous placement method on circuit `ibm04` from the ICCAD04 mixed-size placement benchmark suite; (a) initial result after the first quadratic placement; (b) roughly one third through placement; (c) roughly two thirds through placement; and, (d) prior to legalization and detailed improvement. The fairly non-overlapping placement prior to legalization is obtained without the use of cutlines and partitioning.

but rely on partitioning the placement area to remove cell overlap in a top-down fashion. Force-directed methods, however, do not use partitioning, but rather eliminate cell overlap through the introduction of additional forces. As such, force-directed placement methods are typically used in conjunction with a *legalization strategy* to purge any remaining overlaps after global placement (cf. Section 2.2.2).

The solution to the quadratic optimization problem in (2.1.4.1) results in a cell placement with significant overlap. For example, Figure 2.5 (a) shows the placement for circuit `ibm04` from the ICCAD04 mixed-size placement benchmark suite [2] after the solution of an unconstrained quadratic program—significant cell overlap is present.

To deal with the problem of cell overlap, `Kraftwerk` [46] and subsequently `FDP` [128] apply additional *constant* forces to distribute cells evenly throughout the placement area to reduce cell overlap. The quadratic equation (2.1.4.1) is extended with an additional constant force vector $\mathbf{f}_x$ yielding

$$\mathbf{Q}_x\mathbf{x} + \mathbf{c}_x + \mathbf{f}_x = \mathbf{0}. \tag{2.8}$$

The vector $\mathbf{f}_x$ is used to perturb the placement in the x-direction such that cell overlap is reduced (a similar optimization can be performed in the y-direction). It is easy to show that the additional forces do not restrict the solution space and that any given placement can satisfy (2.8) by proper selection of $\mathbf{f}_x$ [46].

This force-directed approach is iterative; cell overlap is not removed just by solving a single instance of (2.8). Instead, the cell overlap is slowly removed over numerous iterations with the additional constant forces being updated at each iteration to reflect the changing distribution of cells throughout the placement area. Hence, the additional constant forces are *accumulated* over iterations and the force equation at any given iteration i can be written as

$$\mathbf{Q}_x\mathbf{x}_i + \mathbf{c}_x + \sum_{k=1}^{i-1} \mathbf{f}_x^k + \mathbf{f}_x^i = \mathbf{0}. \tag{2.9}$$

The additional constant force is divided into two parts, namely those forces accumulated over previous placement iterations 1 through $i-1$ and a current constant force computed at iteration i. Equivalently, the additional constant force computed at any given iteration is broken into *two specific components*, namely (a) a *stabilizing force* that holds the current placement in equilibrium (represented by the accumulation of forces from previous iterations) and (b) a *a perturbing force* computed for a given placement to further reduce cell overlap.

Numerous other techniques have been proposed for augmenting the quadratic placement formulation to remove cell overlap. These methods include the concept of fixed points, in `ARP` [47], `mFAR` [59], and newer versions of `Kraftwerk` [108], bin shifting in `FastPlace` [120], and

frequency-based methods employing a discrete cosine transform in `UPlace` [29, 136]. The reader is referred to [71] for more details of these techniques.

More linearized forms of the HPWL objective have also been studied within the context of analytic placement. In the patent by Naylor et al. [93], the HPWL of a hyperedge is approximated using a log-sum-exp formula, given by

$$\mathrm{HPWL}_\lambda(e) = \alpha \left(\ln\Big(\sum_{v_j \in e_i} e^{\frac{x_j}{\alpha}}\Big) + \ln\Big(\sum_{v_j \in e_i} e^{\frac{-x_j}{\alpha}}\Big) + \Big(\ln\Big(\sum_{v_j \in e_i} e^{\frac{y_j}{\alpha}}\Big) + \ln\Big(\sum_{v_j \in e_i} e^{\frac{-y_j}{\alpha}}\Big) \right) \tag{2.10}$$

where α is defined as a "smoothing parameter". The smaller the value of α, the more accurate the approximation to (2.1). However, α cannot be chosen to be too small due to machine precision and numerical stability. In effect, the use of the log-sum-exp formula picks the dominant cell positions to approximate the exact HPWL for each edge as specified in (2.1). Despite its use of transcendental functions, the approximation in (2.10) is both differentiable and strictly convex which makes it relatively simple to minimize.

To spread cells, it is desirable to augment the log-sum-exp form with a penalty function that penalizes the uneven distribution of cells. To this end, `APlace` [64–66, 93] imposes a grid on the placement area and attempts to equalize the total cell area in every grid bin. `APlace` approximates the total cell area in each grid bin by "area potentials" for each cell. The area potential uses a bell-shaped function to model the effect of a cell's area on nearby grid bins. This function enables one to represent a continuous penalty term which is combined with the log-sum-exp approximation to arrive at a linearly-weighted objective function representing a trade-off in linear wire length minimization and the quadratic overlap penalty.

`APlace` is an example of a continuous placement method that deviates from the traditional methods such as `Kraftwerk`. In particular, no component of `APlace` necessarily has a direct analogy with the concept of a "force". Hence, its relationship to other force-directed methods is limited to its removal of cell overlap without the need to partition the placement area and, perhaps, its use of the conjugate gradient method for minimization—it is reasonable to interpret the gradient of the objective function used in `APlace` as a "force" which specifies a direction for cell movements.

Alternatively, while `Kraftwerk` and its descendants spread cells using a Poisson distribution, a mathematical model based on the Helmholtz equation is employed in `mPL` [26]. This method is shown to be a generalization of `Kraftwerk` [71], and the reader is referred to [71] for more details.

2.2 Placement Techniques Pertinent to Specific Design Styles

This thesis deals primarily with the improvement of stochastic search techniques across several different fields of VLSI CAD—FPGA placement, mixed-size circuit legalization, and standard cell detailed placement. While the thesis aims to improve search-based methods in each of these cases through the introduction of directed moves, it is nevertheless important to discuss the relevant advances in each field in order to be able to establish a basis for fair comparison.

2.2.1 FPGA Placement

2.2.1.1 Simulated Annealing and VPR

Simulated annealing [75] is an important method for placement in FPGAs due to its flexibility in handling complex objective functions and constraints.

VPR [19] is, perhaps, the best-known annealing-based, academic FPGA placer. Through VPR, Betz et al. introduced several key improvements to FPGA placement, including the concept of path-based weighting for timing-driven optimization, timing-driven clustering, fast incremental bounding box computation, and a well-described, experimentally-tuned annealing schedule.

Timing-driven VPR employs the following delta-cost function:

$$\Delta C = \lambda \cdot \frac{\Delta C_T}{\text{Previous } C_T} + (1-\lambda) \cdot \frac{\Delta C_W}{\text{Previous } C_W} \tag{2.11}$$

where C_T is the timing cost, C_W is the wire length cost, and λ represents the trade-off (that is, "timing_tradeoff") between wire length and timing minimization. The value of C_T is computed based on the timing cost $C_{T_{ij}}$ for each edge (i, j) in the circuit timing graph, and can be expressed as:

$$C(i,j) = 1 - \frac{\text{Slack}(i,j)}{\text{Delay}_{\max}} \tag{2.12}$$

$$C_{T_{ij}} = \text{Delay}(i,j) \cdot C(i,j)^{\beta} \tag{2.13}$$

$$C_T = \sum_{\forall \text{ edges } i,j} C_{T_{ij}} \tag{2.14}$$

Here, β is used to weight connections that are more critical, and increases slowly (starting from 1) over the course of the anneal to a maximum value. VPR recomputes the circuit timing at every temperature change, but only recomputes changes in delays within the inner loop of the anneal (at the same temperature). This allows it to achieve good run-time while still maintaining a reasonably accurate view of timing.

The value of C_W is computed using a weighted bounding box where the weights compensate for

the fact that the bounding box wire length model underestimates the wiring necessary to connect nets with more than three terminals [19]. The bounding box is also weighted in the x and y directions to penalize placements which require more routing in areas of the FPGA with narrower channels; although, in practice, most academic CAD investigations (including those in this thesis) employ architectures where the channels have similar capacities. In practice, the wire length cost function can be quickly computed using an incremental bounding box technique [19].

The annealing schedule employed by VPR has proven to be very successful. As described in [112], the maximum number of moves evaluated at each temperature is given by:

$$\texttt{inner\_num} \cdot N^{\texttt{inner\_exp}} \tag{2.15}$$

where N represents the number of movable logic cells. Note that inner_exp is equal to 1.33 and inner_num is set to 10 by default.

The temperature reduction strategy in VPR follows from the reasoning that, at high temperatures, the anneal is oscillating randomly from one placement to another and little improvement in cost is obtained, while at low temperatures, too few moves are accepted [19]. Consequently, the temperature update schedule employed in VPR follows the form $T_{\text{new}} = \alpha \cdot T_{\text{old}}$ where α is a parameter that varies between 0.5 and 0.95 depending on the annealing success rate.

VPR strives to keep the success rate of its random pairwise moves around 0.44, and does so by varying the size of the random window. The range of the random window is initially set to the size of the entire chip, and is gradually shrunk over the course of the anneal, as the success rate declines. The anneal terminates when the temperature is less than a small fraction of the average cost of a net [19].

2.2.1.2 Other Approaches to Improving FPGA Placement

Since the inception of VPR, numerous attempts have been made to improve upon placement quality and run-time. Most techniques have fallen into one of the following categories: (1) better initial placement (e.g., by coupling annealing with another placement strategy), (2) better clustering (for clustered architectures), (3) modifications to the circuit netlist (e.g., through logic duplication or basic logic element [BLE]-level placement), (4) more accurate objective functions (e.g., path-based timing weights, or the incorporation of congestion metrics), or (5) changes to the annealing schedule (temperature, acceptance criteria, and so on).[5]

Alternative strategies have been investigated to reduce the amount of run-time spent in high-

5 Some of these schemes are only applicable to clustered architectures, although fine-grained, non-clustered architectures also exist; cf. [1].

temperature annealing regimes. For instance, faster placement heuristics, such as recursive min-cut partitioning, have been employed [83, 116] to quickly produce better initial placements, thereby requiring the annealer to run only in a lower temperature regime.

Better packing algorithms have also been shown to improve the quality of annealing-based placements for clustered architectures. A new method for packing logic was proposed in `HDPack` [30] where physical cell distances were incorporated into the packing cost function. A routability-driven packing algorithm was described in `iRAC` [105] where careful attention was paid to the selection of the seed BLE and the packing was controlled by the Rent parameter of the architecture. Depth-optimal and depth-relaxed packing methods [39, 43, 114] have also been described in which timing-critical logic was duplicated during clustering to obtain a set of clusters with optimal depth.

Recent approaches for improving quality during annealing-based placement have focused on modifying the circuit netlist prior to, during, or after placement. One method has been to perform logic duplication and path straightening [17, 31, 32, 57]. Alternatively, a clustered (combinational logic block, or CLB) netlist itself can be re-clustered during placement by moving BLEs, as performed in `SCPlace` [31].

The design of better annealing objective functions has also been considered. By changing timing weights [138] and deriving more accurate models for net costs, better placement results have been reported. For instance, [77] describes a method for computing timing criticalities based on path counting and the use of a discount function. Numerous works have also discussed the integration of global routing and placement to improve the fidelity of the placer's objective function [103].

Run-time improvements have also been discussed in the recent literature. For instance, parallel techniques have been described to accelerate annealing [25, 81]. Shorter temperature schedules have additionally been proposed [119].

2.2.2 Mixed-Size Legalization

While force-directed methods have become a popular choice for the global placement of standard cell and mixed-size designs, they have heightened the need for more effective solutions to the *legalization problem*, in which module overlaps must be removed after global placement. This problem is amplified in force-based methods because placements produced by such techniques generally possess more overlap than that found in constructive approaches such as recursive partitioning. (Figure 2.5 (d) illustrates the need for special techniques to purge the final overlaps from designs produced by force-based methods.)

The legalization of modern mixed-size designs is extremely difficult due to the presence of

both macrocells and standard cells. Typically, the scale of these problems is such that optimal approaches (such as those employed in [74, 87, 111]) are impractical and heuristics are required to arrive at answers in reasonable run-times. Recent trends in the VLSI literature have attempted to address the issue of legalizing mixed-size designs by borrowing concepts from the floorplanning literature, as well as by introducing new heuristic methods. Modern legalization techniques aim to perturb the placements as little as possible in order to maintain the quality of the global placement, although simultaneous optimization of other objectives is sometimes undertaken.

Annealing is a popular approach to solving the floorplanning problem. Such floorplanners typically permute sequence pairs [3, 4, 89] or edges in constraint graphs [80] to attain a feasible, non-overlapping placement. These methods can achieve high-quality, legal floorplans for small problem instances; however, they may not produce optimal results and can suffer from exponential run-time scaling due to the size of the search space in larger problems.

Greedy heuristics account for much of the literature in large-scale, post-placement VLSI legalization. A greedy shifting heuristic for simultaneously legalizing macrocells and standard cells was proposed in [8], based on the patent of [56]. In this approach, a front-end contour (which designates the leftmost empty site on each row) is maintained. Movable cells are traversed in y, and the location of each cell (along the contour) is determined by considering the resultant wire length and displacement penalty. This technique often results in illegal placements with cells extending beyond the chip; thus, cells are typically packed very tightly [8] or larger blocks may be placed first [64] to compensate.

An approach combining a transportation problem and cell rippling was presented in [20]. This work focused on minimum-movement legalization of standard cell designs. In this method, a grid is built over the placement area and buckets are connected to the flow source nodes by edges with flow equal to the amount of overlap in each bucket. The buckets themselves are connected in each direction, and buckets without overlap are connected to flow sink nodes. Buckets possess "soft" vertical boundaries to reduce the flow and therefore reduce the cell movement. The solution to the flow problem yields the edges along which cells should be rippled to minimize overlap.

In [90], a two-step approach is employed to remove overlap with minimal movement in FPGA placements. In the first step, a topological constraint graph [80] is built from the macrocells by means of a sequence pair [89]. Slacks are computed on the constraint graph using a "timing analysis", and critical arcs in the graph are improved by permuting the sequence pair accordingly. Once the constraint graphs are legal, the large cells are placed by allocating the slacks in the constraint graph using a linear-time heuristic. The large cells are subsequently fixed in place, and the smaller cells are positioned via bipartite matching.

Doll et al. [44] divide the placement into multiple regions, and optimize each region by solving a transportation problem. The algorithm allows regions to overlap in order to avoid becoming

trapped in local optima. In [61], Hur and Lillis employ a minimum-distance cell rippling strategy to move cells from regions of high occupancy to regions with free capacity.

In `XDP` [40], a constraint graph (along the lines of [90]) is used to legalize macrocells with a Tetris-like shifting [8] employed to legalize the standard cells. `XDP`'s standard cell shifting employs both a front *and* a back-end contour. Standard cells are moved to the site between the two contours which gives the shortest wire length. Macrocells are considered for movement between the interval determined by the two contours, and the contours are updated appropriately.

`Floorist` [86] addresses the legalization problem by solving a constraint satisfaction problem for macrocells. A pair of constraint graphs is built to model the overlapping features, and a greedy search (guided by critical paths) is employed to render the graphs legal. A final stage translates the constraint graphs into a coordinate representation. Standard cells are then legalized using `Capo`'s traditional snap-to-site approach [2].

2.2.3 Detailed Placement

Detailed placement is performed in standard cell designs after both global placement and legalization. Generally-speaking, the goal of detailed placement is to:

- optimize objectives which were not handled during global placement;
- undo any harm to the placement caused by legalization; and,
- improve upon the final quality of the placement.

Within the context of detailed placement, both heuristic and optimal strategies have been explored.

The prevailing methodologies for detailed placement in standard cell designs rely on optimal techniques which can be employed on small, localized subsets of cells, although search-based techniques have also been used in practice. Since modern problems can possess millions of standard cells, strict mathematical programming formulations are not usually favoured; rather, most attention in the literature has focused on one of three generic methods: stochastic search, branch-and-bound rearrangement, and flow-based strategies.

2.2.3.1 Stochastic Search Techniques for Detailed Placement

Historically, simulated annealing has been viewed as a poorly-scalable, time-consuming strategy for detailed placement, and it is for this reason that modern detailed placers generally focus on alternative strategies. Some burden for this belief may be borne by `TimberWolf` [101], which employed simulated annealing for both global *and* detailed placement, leading some to conclude that the technique's run-time scalability was poor.

The cost function originally employed by `TimberWolf` [101] accounts for wire length, but also penalizes module overlap and the length of standard cell rows. This cost function can be described as

$$\phi = \sum_{\#\text{ nets}} (\alpha_x \text{bb}_x(i) + \alpha_y \text{bb}_y(i)) + \alpha_o \sum_{i \neq j} \left(\text{OL}(i,j)^2\right) + \alpha_r \sum_{\#\text{ rows}} |\text{ARL}(i) - \text{DRL}(i)|\,.$$

Here, bb_x and bb_y denote the horizontal and vertical spans of net i's bounding box. α_x and α_y are weights applied to the horizontal and vertical wiring spans. The function $\text{OL}(i,j)$ calculates the amount of overlap between cells i and j, while α_o acts as a weighting for the overlap penalty. The quadratic nature of this overlap term discourages large overlaps. The third term of the objective equalizes row lengths by increasing the cost if rows are unequal lengths. In this case, $\text{ARL}(i)$ and $\text{DRL}(i)$ represent the actual row length and the desired row length of row i, while α_r allows the term to be weighted appropriately.

`Timberwolf` chooses cells randomly and either interchanges or displaces these cells to a random location in the chip. The algorithm performs best when the number of displacements is between three and eight times the number of interchanges [101]. In more recent versions of `Timberwolf` [110], cell overlap is only briefly permitted when an interchange or displacement is performed, but any resulting overlaps are purged by shifting cells to the left or right. Nevertheless, because the effects of this shifting are not precisely modelled, the cost function used in [110] does not exactly model the HPWL of the design.

Stochastic search techniques have been mentioned in more recent works, but only in limited contexts, such as greedy (zero-temperature) approaches. In `mPL` [27], the authors state that window-based cell swapping is employed to reduce wire length, that "all the cell permutations within [a] window are examined", and that "the [permutation] giving the shortest scaled wire length is accepted" [27]. It is unclear, from the work, what types of cells are swapped (whether they must be similarly-sized or not), what types of strategies are employed to maintain legality, how much improvement is achieved, and whether or not this is done within an annealing context. Greedy swap-based methods are also mentioned in [64, 95].

2.2.3.2 Small-Window Detailed Placement

As a result of the questionable success of annealing-based techniques, the focus for detailed placement shifted to optimal arrangement of small subsets of cells. Absent any whitespace, placement solutions can be considered as permutations of hypergraph nodes [23]. The detailed improvement problem, then, lends itself to enumerative strategies which employ branch-and-bound techniques to prune the search space.

Branch-and-bound placement in a single dimensions (i.e., a single row of a standard cell circuit) has been well documented in the literature [23]. In this method, a window is scanned over each row,

and cells are "ripped up" in subsets of at most 8 (or so) cells. Cells are then added to the placement problem one at a time, and the bounding boxes of incident edges are extended to consider the new locations of each cell. From a given partial placement, the *lower bound* of the wire length of any completion of the placement is computed. Extensions of the current partial solution are considered only as long as this lower bound is smaller than the cost of the best seen complete solution [23].

To accommodate varying dimensions, cells may be packed with a fixed-size space between neighbours—in other words, whitespace is distributed equally between them. Alternatively, it may be possible to consider portions of whitespace themselves as dummy cells in the problem, and optimize their location when placing logic cells. Replacing a cell with a cell of a different width changes the location of at least one neighbour, and triggers bounding box re-computations for incident nets [23].

Typically, cells are packed from left to right and are always added or removed (using a lexicographic ordering) from the right end of the partially-specified permutation, as shown in Figure 2.6. This formulation lends itself to a stack-based implementation where the states of incident nets are pushed onto stacks when a node is appended on the right side of the ordering, and popped when the node is removed [23]. Bounding is performed by removing cells from the end of a partial solution before all lexicographically greater partial solutions have been visited [23]. Pseudocode for this procedure is provided in Figure 2.7.

Multi-row branch-and-bound placement techniques have received some attention in the literature. In [97], a mixed-integer linear program is presented to model the placement of unit-sized standard cells in windows of up to 6×6 cells. Due to the complexity of the problem and the absence of good bounding criteria, the run-times for such a technique are impractical.

Dynamic programming techniques have also been employed for standard cell optimization. For instance, a cell assignment technique is considered in `NTUplace3` [33] in which an assignment problem is employed to place up to 200 to 300 unrelated cells (within a window) at a time.[6] A graph colouring problem is used to gather the unrelated cells. Rather than using a min-cost max-flow solver, however, the implementation in [33] solves a weighted bipartite matching problem, which is conceptually similar though potentially more run-time efficient. Hur and Lillis [61] proposed a detailed placement algorithm based on dynamic programming which iteratively picks two groups of cells in the placement and optimally interleaves them.

2.2.3.3 Congestion Control

While the optimization of HPWL has been considered extensively in the literature, some work has also focused on routing congestion minimization. By minimizing routing congestion, a placer can

6 This work bears a significant resemblance to the `Domino` [44] approach described in Section 3.3.2.1.

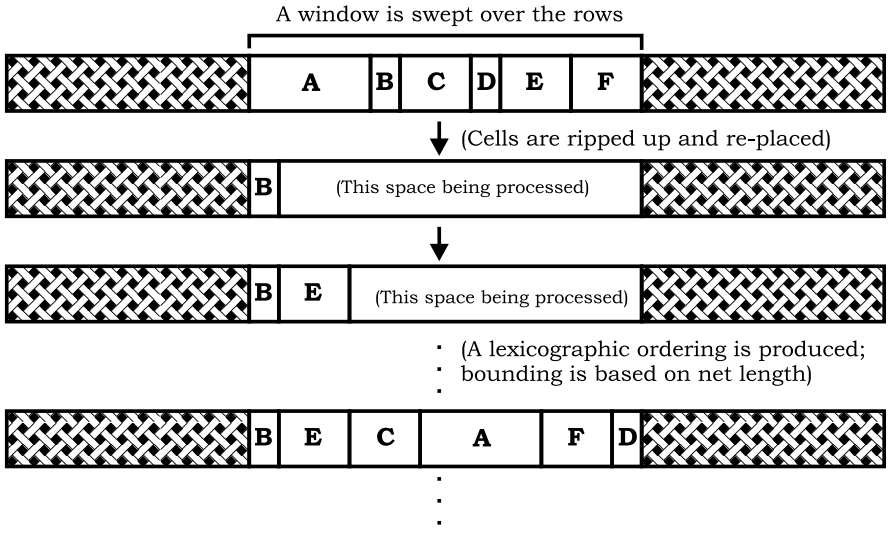


Figure 2.6: Illustration of single-row branch-and-bound placement.

improve the routability of a design by ensuring that the routing demand across a chip is less than or equal to routing supply.

Routing congestion in a mixed-size ASIC design is most often modelled by overlaying a grid on top of the placement. The congestion is related to the number of routed nets crossing the edges of the individual grid bins. This formulation serves to reduce the computational effort, while still affording the technique a reasonably precise view of routing demand. Several techniques—such as those based on global routing and Rent's Rule [58, 131, 135]—have been employed to give a more accurate idea of the amount of "net overflow" leading into these congestion maps.

The ISPD 2006 placement contest [91] incorporated a modified objective function which accounts for both wire length and a "scaled overflow" metric. This overflow metric approximates the notion of routability-driven optimization by penalizing dense placements. The ISPD 2006 cost function can be expressed as:

$$\min \Phi = \text{HPWL} \times (1 + \text{scaled\_overflow}). \tag{2.16}$$

The scaled_overflow term approximates the "congestion" in a design, and is computed by imposing a uniform grid on top of the placement, calculating the utilization for each grid bin (where utilization is given by the area of movable objects divided by the capacity of the bin), and summing the utilization for all bins. The overflow term is then squared and scaled linearly to be within limits set by the contest.

Congestion can be alleviated by inserting whitespace after placement according to routability metrics or other parameters (cf. [98, 99]). However, whitespace can also be crudely handled by

```
Procedure: SINGLE-ROW B&B
Input: A queue of relatively non-overlapping cells to place, s
Variables: A queue, q, which will contain the current working set
1  begin
2    idx ← numCells −1;
3    costSoFar ← 0;
4    bestCostSeen ← cost of current placement;
5    nextLocation ← set to the sub-problem's leftmost edge;
6    while idx < numCells do
7      s.push( q.deque() );
8      cnt[idx] ← idx;
9      costSoFar ← costSoFar + cost of placing s.top();
10     nextLocation = nextLocation + widthOfCell( s.top() );
11     if costSoFar ≤ bestCostSeen then
12       cnt[idx] ← 0; //This signals a bounding.
13     fi
14     if cnt[idx] = 0 then
15       //The ordering is complete or has been bounded.
16       if idx = 0 and costSoFar < bestCostSeen then
17         bestCostSeen ← costSoFar;
18         save current placement;
19       fi
20       while cnt[idx] = 0 do
21         //Remove the right-most cell.
22         costSoFar ← costSoFar - cost of placing s.top();
23         nextLocation ← nextLocation - widthOfCell( s.top() );
24         q.enqueue( s.pop() );
25         idx ← idx + 1;
26         cnt[idx] ← cnt[idx] - 1;
27       od
28     fi
29     idx ← idx - 1;
30   od
31 end
```

Figure 2.7: Pseudocode for a single-row branch-and-bound placement algorithm. The placement is produced from a lexicographic enumeration, and bounding is based on the best cost seen.

"bloating" cells during placement and then restoring cell dimensions afterwards.

Although global placers (and possibly legalizers based on minimum-movement concepts) aim to satisfy density targets (if provided), careful consideration to bin overflow must be paid during detailed placement as well. For example, in `NTUplace3`, cells in over-utilized bins are shifted left and right during detailed placement until the over-utilization is minimized—cells are shifted from denser bins to sparser bins while preserving cell ordering. Only horizontal sliding is considered, as vertical sliding was found to produce misalignment between standard cells and site rows [33]. This post-placement approach to congestion minimization can have deleterious effects on HPWL.

Several strategies for whitespace allocation were considered in the partitioning-driven placer, `Capo` [100]. For example, whitespace can be distributed uniformly by proper cutline management during partitioning. Such whitespace allocation "generally produces routable placements, at the cost of increased wire length" [100]. Local tolerances can also be adjusted to control the shifting of the cutline to produce less-uniform whitespace allocation, which can allow for tighter-packed placements and better wire length quality.

CHAPTER 3

Improving Simulated Annealing for FPGAs with Directed Moves

3.1 Overview

This chapter investigates the concept of "directed moves" as a means of improving simulated annealing-based FPGA placement. This notion was inspired by previous research in the ASIC domain [4, 55] where intelligent, deterministic strategies for selecting and replacing "poorly-placed" cells were *interspersed* with simple, random moves during annealing. Directed moves serve to reduce the size of the search space by focusing on cells and locations of interest; this allows the annealer to converge more quickly and to attain a better placement for the same amount of run-time as if random moves had been used alone.[1]

In this chapter, several concepts for directed moves are considered, such as median placement [52,128], cell rippling, graph colouring, optimal linear assignment [44], and the minimization of monotone path deviation [17, 32, 57]. These directed moves were implemented in a modern, C++-based academic FPGA placement framework called `KPF`. The results confirm that, for the same amount of computational effort, wire length-driven and timing-driven directed moves—when used in combination with simple moves—routinely lead to improvement in both critical path delay and wire length, compared to having used simple moves alone. Moreover, directed moves are shown to reduce the statistical variability of the annealing-based placements. Additionally, a

1 Portions of this chapter were published in [126, 127].

technique is presented for measuring the effectiveness of moves, and for terminating the anneal. Further numerical results explore the effects of moves in clustered FPGA architectures.

Section 3.2 motivates the concept of directed moves for FPGA placement. Section 3.3 discusses the implementation of the directed moves. Finally, Section 3.4 presents numerical results.

3.2 Motivation for Directed Moves

The motivation for this chapter—and, to a larger extent, this thesis—stems from the observation that an annealer may spend time re-visiting previously-explored states, and may waste run-time before discovering the lowest-cost states. In simulated annealing, the function for generating a new state j given a current state i is given as $g(i, j)$. A matrix $\mathbf{P}$ can be used to represent the state transition probability of a traditional annealing process [75], and takes the form:

$$P_{ij}(T) = \begin{cases} 0 & \text{if } j \notin N(i) \text{ and } i \neq j \\ \min\{1, e^{-\frac{F(j)-F(i)}{T}}\} & \text{if } j \in N(i) \\ 1 - \sum_{j \in N(i)} P_{ij}(T) & \text{if } i = j \end{cases} \tag{3.1}$$

where $N(i)$ represents the neighbours of the state i, $i \notin N(i)$, T is the temperature, and $F(i)$ is the value of the cost function in state i. For a given T, the probability distribution of the annealing problem can be viewed as a stationary time-homogeneous Markov chain, represented as:

$$\pi_i(T) = \frac{g(i)e^{-\frac{F(i)}{T}}}{G(T)} \tag{3.2}$$

where $g(i)$ is a normalizing function such that $\sum_{j \in N(i)} \frac{g(i,j)}{g(i)} = 1$, and $G(T)$ is a scaling factor such that $||\pi(T)|| = \sum_{i=1} |S|\pi_i(T)$.

For this Markov model to converge to the optimal solution, the neighbour generation criterion must satisfy $g(i, j) = g(j, i)$ for all states i, j [85]. This leads to the stationary probability condition $\pi(T)\mathbf{P}(T) = \pi(T)$, which in turn, leads to the detailed balance criterion:

$$\frac{\pi_i(T)}{\pi_j(T)} = \frac{g(i)}{g(j)} e^{(-\frac{F(j)-F(i)}{T})} = \frac{P_{ji}(T)}{P_{ij}(T)}. \tag{3.3}$$

That is, the probability of changing from state i to state j must be the same as the probability of generating i when the system is in j times the probability of accepting it [102].

If $g(i, j)$ could be made to explore neighbour states that are more likely to yield improvement, the amount of time required for the anneal can be reduced. This preferential state exploration forms the basis for the directed moves described in this work.

If directed moves are implemented without consideration for the detailed balance criterion, the risk of oscillation and of converging to a local minimum is raised. It is worth noting that it is possible to implement directed moves which do not harm detailed balance by ensuring that the probability of accepting the "reverse" move (i.e., P_{ji}) compensates for the earlier, directed state exploration. However, as will be shown in this work, this compensation is not necessary, in practice, to achieve high-quality placements provided that a sufficient portion of attempted moves are still of the traditional, non-directed kind.[2] Moreover, directed moves do not preclude the attainment of high-quality results in a practical annealer since numerous heuristics (such as clustering, windowing, non-infinite starting temperatures, and so forth) are already employed to reduce run-time at the expense of producing sub-optimal solutions. Thus, the goal, in this work, is for directed moves to serve as a means of "shifting" the cost function curve so that better-quality placements can be produced more quickly.

Only two prior works— [4, 55]—are known to have discussed the concept of directed moves in the context of cell placement. In [4], two types of moves are considered in the `Parquet` ASIC floorplanner: a source-selection strategy to increase horizontal and vertical slacks in the floorplanning problem, and a target-selection method for moving cells toward wire length-reducing locations. The source-selection strategy is performed using a "priority list" approach in which `Parquet` performs operations more frequently on the cells with worse slacks. The wire length move is computed by occasionally solving a quadratic optimization problem and then trying to move a given block closest to the average of the position of all modules connected to it. Alternatively, in [55], a move based on weighted centroids was interspersed with random moves to improve the quality of the wire lengths in the `Timberwolf` [101, 110, 113] standard cell placer. The "schedule" for determining which moves to choose and how often to perform them, in both cases, is hard-coded.

This work presents five additional contributions for directed moves compared to the previous body of literature:

1. many new types of directed moves are considered;

2. a new technique (based on cell rippling) to retain placement feasibility is described;

3. the use of multiple directed moves to optimize separate annealing objectives (such as both wire length and critical path delay) is explored;

4. a heuristic for automatically ranking moves and for determining when to terminate an anneal based on the moves' effectiveness is presented; and,

2 The move selection strategy described in Section 3.3.4 ensures that this is the case.

5. a comprehensive quality versus run-time analysis of the directed moves is discussed for both clustered and non-clustered FPGA architectures.

While some of the directed moves described in this chapter have been employed in post-placement optimization strategies (e.g., [17]), the integration of these methods into the annealing loop (1) unifies and simplifies the implementation and (2) allows the methods to consider a wider search space which can lead to potentially better results.

3.3 Implementation

An academic annealing-based placer, called KPF, was developed as part of this work. It targets the same type of architecture and CAD flow as VPR.

In KPF, a "move" is the fundamental operation of perturbing a placement (i.e., generating a neighbour state)—picking a set of source ("from") cells, S, choosing a set of target ("to") locations, T, and then assigning S to T. Like most annealers, feasibility is maintained throughout the anneal. Thus, if a cell S_0 is assigned to an occupied location T_1 (i.e., occupied by cell S_1), it will either assign S_1 to T_0 (as in VPR) or *ripple* a set of cells along a path from T_1 to T_0 to retain legality. The concept of rippling will be discussed in more detail in the following sections.

In KPF, multiple cells can be assigned to multiple locations in a single perturbation—the delta cost for moving all cells is computed at once and the *entire move* is either accepted or rejected. It is possible that significant improvement could be wrought from such a move, but at the same time, if the move is rejected, it could be costly in terms of run-time.

VPR uses a random strategy to choose *source* cells and a random, shrinking window method to choose *target* locations; these moves are referred to as "simple moves". KPF implements such moves as well as other strategies for picking source cells and target locations. In addition, both assignment and rippling have been investigated as means of retaining feasibility, as summarized in Table 3.1. As discussed in Section 3.4.1, not all possible combinations of strategies in this table were exhaustively tested, and of the moves which were tested, not all proved to be useful. This work focuses primarily on the subset of moves which did yield improvement in early testing.

3.3.1 Heuristics for Determining Source Cells

Three heuristics were implemented for determining source cells. The first heuristic was a traditional **random** cell selection, as implemented in VPR.

The second heuristic was based on the concept of **graph colouring** in which the netlist hypergraph is coloured *prior* to placement. During placement, the colouring is used to randomly choose independent, non-connected cells (i.e., cells which do not share a net) in subsets of up

Table 3.1: Combinations of source and target cell moves considered in this chapter, as well as the methods considered for resolving infeasibilities (via *assignment* or cell *rippling*). For comparison purposes, "**WL**" or "**CP**" is used to indicate whether the move targets primarily wire length or critical path delay. "♡" is used to indicate moves that lead to higher quality and "♃" to designate higher run-time complexity.

Target Selection ⇒ / **Source Selection ⇓**	**Random**	**Domino** (WL)	**Median** (WL)	**MPD** (CP)	**Weighted Median** (CP)	**Centroid** (CP)	**Priority List** (CP)
Random	Assign (♡♡♃)	n/a	Assign (♡♡♡♃♃) Ripple (♡♡♡♡♃♃♃)	Assign (♡♡♡♡♃♃) Ripple (♡♡♃♃♃)	Assign (♡♡♃♃♃) Ripple (♡♡♡♃♃♃♃)	Assign (♃)	Assign (♡♃)
Colouring	n/a	Assign (♡♡♡♡♃♃♃♃)	n/a	n/a	n/a	n/a	n/a
Priority List	n/a	n/a	Ripple (♡♡♃♃♃)	Ripple (♃♃)	n/a	n/a	n/a

to 15 cells at a time. This graph colouring permits the later application of the `Domino` technique (described in the next subsection) to compute an assignment (and therefore, a placement) for the cells.

The final selection heuristic was based on **priority lists**, where a cell is chosen, at random, from a list containing the 25% *worst-placed* cells in the design. At each annealing temperature update, this priority list is recreated by scoring the cells based on (1) their distance away from their optimal half-perimeter wire length (HPWL) positions as well as (2) the maximum timing cost of paths through the cell. The goal of this source selection strategy is to "focus" the efforts of the annealer so that it chooses cells that are more likely to yield improvements in quality (and less time is wasted in unproductive moves). The distance from the optimal HPWL positions is measured using the concept of median placement (which is described in the following subsection). The timing cost ranks are computed as a function of pin criticality multiplied by delay for each driver-sink pair.

3.3.2 Heuristics for Determining Target Locations

Several heuristics for determining the target locations for cells were implemented. These methods generally incorporate aspects of randomness within a "focused" area, and are usually geared toward optimizing wire length or timing. The first heuristic was a standard, **random** target location within a shrinking window, as implemented in [19]. Additional, more complex moves are considered below.

3.3.2.1 Domino

In [44], the authors formulate a minimum-cost flow problem for determining how to place subsets of approximately 20 standard cells. The algorithm, entitled `Domino`, works by first "binning" the placement area with a set of overlapping bins—cells are "snapped" into their nearest bins. (Some cells may be assigned to multiple bins based on how the bins overlap.) `Domino` "shreds" large cells (to accommodate varying standard cell dimensions) by turning them into groups of smaller cells connected by high edge affinities. Valid locations for the cells are then determined prior to cell assignment.

Each sub-problem is solved using a transportation problem that moves cells to locations such that the transportation cost is minimized. The transportation problem is formulated as a minimal-cost, maximum flow problem as shown in Figure 3.1. The problem consists of a source node S which supplies cells, a set of cell nodes u, location nodes λ, and a sink node, T. The capacities of arcs between node S and each cell node is 1. The cost of assigning cell μ to location λ is given by $c_{\mu\lambda}$. Finally, the cost between locations and the sink node is also set to 1. The solution to the flow problem dictates *how* to move the cells; a heuristic is then employed to *actually* move cells based on where the flow "suggests" they go.[3] After the application of the algorithm, large cells that were shredded are reconstituted at the average locations of their smaller constituents.

The quality of the resultant placement is determined, in large part, by how accurately the cost $c_{\mu\lambda}$ approximates the *actual* cost of assigning a cell μ to location λ. If none of the cells in the sub-problem are interconnected (i.e., no cells share a common net), the cost c can be determined perfectly, and the assignment is optimal. If, however, the cells *do* share common nets, then the assignment is only approximate. This is because the assignment of cell μ to location λ could alter the cost of assigning another interconnected cell to another location.

A method is presented in [44] to help in approximating the costs for cells which share nets. Typically, $c_{\mu\lambda}$ is computed as the HPWL of assigning a cell μ to location λ; that is, given the location (x_λ, y_λ), $c_{\mu\lambda}$ represents the sum of the horizontal and vertical spans of all nets attached to μ if it were placed at (x_λ, y_λ). However, as mentioned above, the cost of nets shared by more than one cell in the sub-problem cannot be accurately computed, since the horizontal and vertical spans of the nets are not known prior to the assignment of the cells. To approximate the cost for such nets, dummy cells ϕ_i are placed at the *centres of gravity* of each shared net. A dummy cell's (x, y) location is determined by

$$x_\phi = \frac{1}{|V(E)|} \sum_{\mu \in V(E)} x_\mu \text{ and } y_\phi = \frac{1}{|V(E)|} \sum_{\mu \in V(E)} y_\mu.$$

3 The heuristic moves cells along the flow such that they end up in non-overlapping positions.

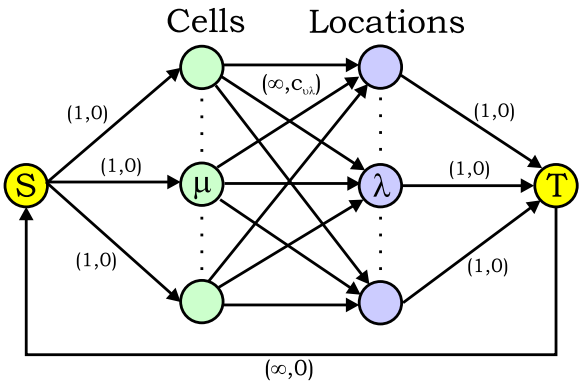


Figure 3.1: Illustration of the transportation problem used in `Domino`. The weights on the arcs denote the capacity and cost, respectively, in the minimum-cost flow formulation.

The contribution of internal net e to the cost $c_{\mu\lambda}$ is then given by:

$$C_\phi = \max(x_\phi, x_\mu) - \min(x_\phi, x_\mu) + \max(y_\phi, y_\mu) - \min(y_\phi, y_\mu).$$

A directed move heuristic was implemented based on `Domino`. Given a set of cells, `KPF` solves a minimum cost linear assignment problem to assign the cells to sites where the sites are located at the positions of the original cells. When setting up the assignment problem, it is important that the costs on each arc closely model the *actual* cost of assigning a cell to a location. By selecting the source cells via graph colouring, it is assured that none of the cells in the transportation problem are interconnected (i.e., no cells share a common net); consequently, the wire length costs on the arcs can be determined perfectly and the placement will be optimal for HPWL [44]. One caveat to this approach is that the assignment ignores timing, and this can result in the move being rejected.

The approach to flow-based improvement presented in this thesis is similar to that of [44]. However, instead of using the flow formulation to determine *how* to move cells, a linear assignment problem is solved to allocate each cell to each same-sized location in every sub-problem. `Domino` uses flow as a guide in determining how to assign cells, whereas the approach presented here uses the same formulation as Figure 3.1 to *actually* make the assignment. A high-performance minimum-cost flow heuristic by Goldberg-Tarjan [36] is used to solve each sub-problem.

3.3.2.2 Median Placement

In [52], Goto proposed an algorithm that can be used to move a cell into a position that minimizes the wire length of its connected nets while assuming that other cells are fixed. The algorithm can be applied iteratively to each cell to obtain an improved placement. Central to the idea of

repositioning a cell is the concept of the *median of a cell*. Goto defines the median of a cell as the position at which the HPWL of its connected nets is minimum.

The median of a cell C is computed as follows. Let E_C denote the set of nets connected to cell C. For each $e \in E_C$, compute the enclosing rectangle of all pins on e while excluding those connections to cell C; the dimensions of this rectangle can be denoted by coordinates (x_e^{min}, y_e^{min}) and (x_e^{max}, y_e^{max}) where x_e^{min} and x_e^{max} are the minimum and maximum values in the x-direction, respectively. The same definitions hold for y_e^{min} and y_e^{max} in the y-direction. Given these definitions, the total wire length for all nets connected to cell C at position (x, y) is given by

$$f_C = \sum_{e \in E_C} (f_e(x) + f_e(y)) \tag{3.4}$$

where

$$f_e(x) = \begin{cases} x_e^{min} - x, & x < x_e^{min} \\ 0, & x_e^{min} \leq x \leq x_e^{max} \\ x - x_e^{max}, & x > x_e^{min} \end{cases} \tag{3.5}$$

$$f_e(y) = \begin{cases} y_e^{min} - y, & y < y_e^{min} \\ 0, & y_e^{min} \leq y \leq y_e^{max} \\ y - y_e^{max}, & y > y_e^{min} \end{cases}. \tag{3.6}$$

Equation (3.4) is separable and the optimal position (x, y) for cell C can be calculated independently in both the x- and y-directions. Goto showed that (3.4) can be written (x-direction only) as

$$f_C = \sum_{e \in E_C} \left[|x - x_e^{min}| + |x - x_e^{max}| \right] \tag{3.7}$$

with the optimal solution given by a median computation. In practice, medians are computed simply by inserting x_e^{min} and x_e^{max} for all $e \in E_C$ into a vector and finding the median value. That is, for a vector of length n indexed 1 to n, a suitable minimizing value for x is any value within the range of values stored at the indices $\lfloor n/2 \rfloor$ and $\lfloor n/2 \rfloor + 1$ of the sorted vector. Figure 3.2 shows the computation of the median rectangle for a cell connected to three nets.

Median placement was used as the basis for a directed move capable of computing target locations in `KPF`. The algorithm follows from Goto's work, with the target location being chosen at random from within the optimal range shown in Figure 3.2.

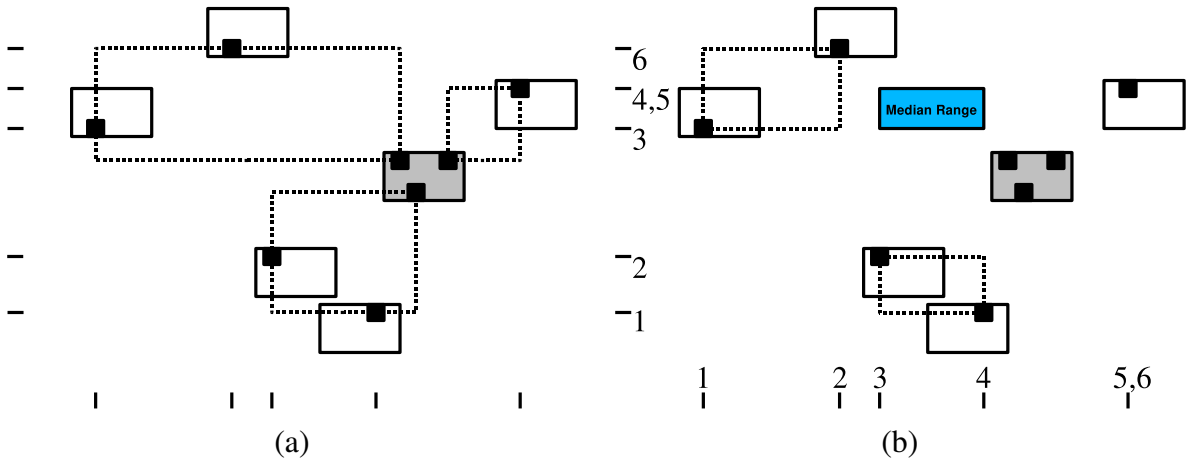


Figure 3.2: Illustration of the median calculation for a cell C connected to three nets. In (a), the original placement of cells is shown. In (b), the median, or optimal range of (x, y) values for cell C is shown. Six x- and y-positions are used for the median computation since three nets are involved. Note that two-pin nets degenerate to a single point. A larger set of positions for cell C can be computed by expanding the median rectangle outward according to the points used in the median computation, thereby implementing the concept of ε-neighbourhoods described by Goto [52].

3.3.2.3 Monotone Path Deviation

To minimize critical path delay, a target heuristic was implemented based on the concept of minimizing **monotone path deviation** (MPD). A monotone path between two nodes is a path with a length that is equal to the Manhattan distance between the nodes [16]. Making the path more monotone has been shown to reduce the delay on the critical path [16], as doing so "straightens" the connections between sources and sinks.

Consider a node n with k inputs, i_k, which are on the critical path. The *monotone region* for node n can be expressed with respect to one of its inputs, i_j, and a node, o, which is driven by n, as the minimum bounding box enclosing i_j, n, and o. Node n can be placed anywhere inside the monotone region without increasing the deviation of n with respect to i_j and o, which allows the subpath $i_j \rightarrow n \rightarrow o$ to be shortened [16, 32]

The key to minimizing monotone deviation, then, is to consider the *intersection* of the monotone regions for the critical inputs of n. This intersection presents a *feasible region* into which n can be placed without increasing the deviation of n with respect to any of the inputs and the output node o. An example of this concept is illustrated in Figure 3.3.

Given a source cell, `KPF` computes the feasible region with respect to the cell's most critical drivers and its most critical sink. The target location for the source cell is subsequently chosen at

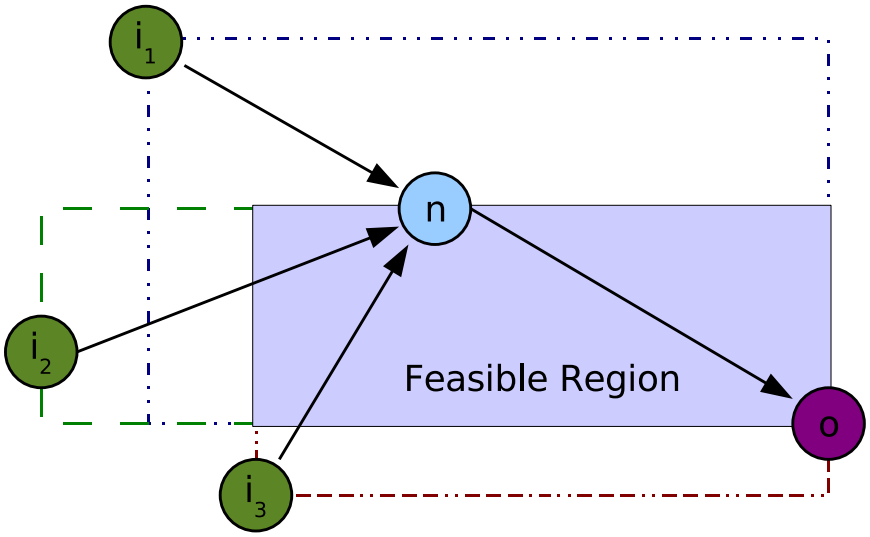


Figure 3.3: Illustration of a feasible region computation for a node with 3 critical inputs and 1 critical output [32].

random from within the feasible region.

3.3.2.4 Priority Lists

Another timing-driven technique for target selection was implemented based on the concept of **priority lists**. In this approach, a list was used to determine target cell locations by tracking the top 25% of cells in the netlist containing the most timing slack. One of the cells in this target list is chosen at random, and the source cell and this newly-chosen cell are swapped. This move is conceptually similar to one employed in `Parquet` [4].

3.3.2.5 Other Approaches

Another modification to the *median placement* strategy was considered where, instead of deriving a region from the median positions of connected cells, a **weighted median** computation is employed to improve timing. In this approach, the median values of the nets are *weighted* by the timing cost of their respective pins. This approach skews the median region toward pins with higher timing costs. The target location into which to move a cell is chosen at random from within the weighted median range. The intent of this heuristic was to produce a move that could simultaneously reduce wire length as well as timing.

Lastly, a **weighted centroid** approach, akin to [55], was investigated. Given a source cell, the target location for the cell is computed as the weighted centroid position of its drivers and receivers,

where the weights are based on the timing costs of the pins.

3.3.3 Resolving Infeasibilities

Early on, it was observed that median placement—despite being optimal for wire length (for a cell moving into the computed median range)—often resulted in numerous rejects when "swapping" with a cell. The median placement would yield good improvement for the first cell which was being moved into its optimal range, but would often result in an increase in wire length for the second cell as it was swapped back into the first cell's original position. To lessen this impact, a new method—cell rippling—was developed to maintain legality during placement.

Cell rippling works as follows. Given a source cell S_0 (at location T_0), some technique (e.g., median placement) is first used to calculate a target location, T_n, for that cell. If T_n is unoccupied, S_0 will simply be assigned to that location; however, if T_n is occupied, S_0 is moved to T_n and the previous contents are *rippled* from T_n by one grid unit toward the nearest empty location (which will be bounded within the distance of T_0 to T_n). This rippling is computed for all cells along the path toward the empty site, creating a set of rippled cells and "spreading" the perturbation across several cells instead of just one cell.

An example of cell rippling is shown in Figure 3.4. In this diagram, cell A was chosen as the source cell and cell B was chosen at random (from within a median region) as its target location. The cell rippling first discovers the nearest empty location within four units of B; B is subsequently rippled *toward* the empty location and placed at C. The cell formerly at C is then rippled to D, and the cell at D is rippled into the empty space at E. Four cells are assigned at one time in this "move" and the entire rearrangement is either accepted or rejected by the annealer.

Although the nearest empty location is always used, the rippling directions are chosen randomly so that in the event of multiple calls with the same source cell and target locations, the chosen rippling path may be different each time. The cell rippling does not attempt to choose a path which minimizes the effect on timing; it was felt that doing so would incur too much of a run-time penalty for the move and might render it too deterministic.

3.3.4 Move Selection and Effectiveness

Empirically, directed moves have been observed to be more effective (relative to one-another) at different times throughout the anneal. Ideally, moves will be employed when they are most likely to improve the cost function. To accomplish this goal, the move selection probabilities are adapted dynamically based on changes in the cost function.

In [62], the effectiveness (or, "quality factor") of moves was computed as part of this dynamic

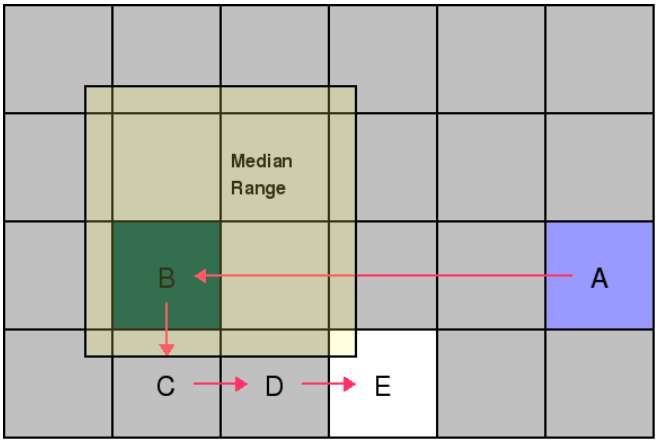


Figure 3.4: Example of cell rippling when used in conjunction with the median placement strategy.

move probability selection. The quality factor for a move type m was computed as follows:

$$Q(m) = \frac{\sum_{\text{accepted moves } i \text{ of type } m} |\Delta c_i|}{\sum_{\text{attempted moves } i \text{ of type } m} t_i} \tag{3.8}$$

where Δc_i is the computed delta cost and t_i is the time taken to compute and perform a move i.

In this work, the effectiveness is computed in a similar fashion, except that the equation is modified as follows:

$$\mathcal{E}(m) = \frac{\sum_{\text{attempted moves } i \text{ of type } m} |p_i \cdot \Delta c_i|}{\sum_{\text{attempted moves } i \text{ of type } m} t_i} \tag{3.9}$$

where p_i is the probability of accepting the move.[4] This reduces the sampling noise.

At the beginning of each temperature change, the annealer uses the relative effectiveness of each move to determine the probability of selecting that move in the forthcoming pass of the annealer. The more effective that a move was in the previous annealing pass, the more likely that it will be chosen in the next pass of the annealer. Specifically, the probability, $\mathcal{P}$, of selecting a move

4 Note that in Metropolis rejection schemes,

$$p_i = \begin{cases} 1 & \text{if } \Delta c_i \leq 0 \\ e^{-\frac{\Delta c_i}{T}} & \text{if } \Delta c_i > 0 \end{cases}.$$

where T is the current temperature.

i is computed as:

$$\mathcal{P}(i) = \frac{\mathcal{E}(i)}{\sum_{\forall \text{moves } j} \mathcal{E}(j)}. \tag{3.10}$$

The probability of selecting a move remains constant until the next temperature change.

To prevent rapid fluctuation in the probabilities from one temperature change to another, the move selection probabilities are smoothed using the function:

$$\mathcal{P}_{\text{smoothed}}(i) = \lambda \cdot \mathcal{P}_{\text{prev}}(i) + (1-\lambda) \cdot \mathcal{P}(i) \tag{3.11}$$

where $\mathcal{P}_{\text{prev}}(i)$ is the probability of selecting move i from the previous annealing temperature pass, λ controls the effect that historical probabilities impart on the selection of current move probabilities, and $\mathcal{P}(i)$ is the probability computed in (3.10). The probability of selecting a move is constrained so that it never falls below a pre-defined threshold—this ensures that poorly-performing moves are occasionally attempted (and not completely eliminated in the event that they experience a "string of bad luck").

Figure 3.5 shows the relative effectiveness of different moves during the placement of a design. In this diagram, three types of moves were employed—median placement (with rippling), MPD (without rippling), and random moves—with the move probabilities of each initially set to 30%, 20%, and 50%. As the temperature cooled, the directed moves became more effective than the random moves. At low temperatures, most cells were placed in their optimal median ranges and timing paths were relatively straight; thus, the directed moves become less effective (for the amount of run-time that was required for their computation) than the simpler random perturbations.

The concept of move effectiveness can also be used to determine when to terminate the anneal. If the condition:

$$\sum_{\forall \text{moves } i} \frac{\mathcal{E}(i)}{\#\text{ attempts for move } i} < \varepsilon \tag{3.12}$$

is satisfied for some user-definable tolerance ε, the annealer can stop because there are no more remaining, effective moves. `VPR`'s criterion [19], on the other hand, is tuned to specific parameters, and is set to stop at an iteration n if $T(n) < \frac{||F(n)||}{200 \times |E|}$ where $T(n)$ is the temperature at n, $||F(n)||$ is the normalized objective function value for the placement, and $|E|$ represents the number of nets in the netlist. `VPR`'s stopping criterion does not work if $T(n)$ is zero, and may not be suitable for all cost functions, whereas move effectiveness is independent of temperature and design parameters.

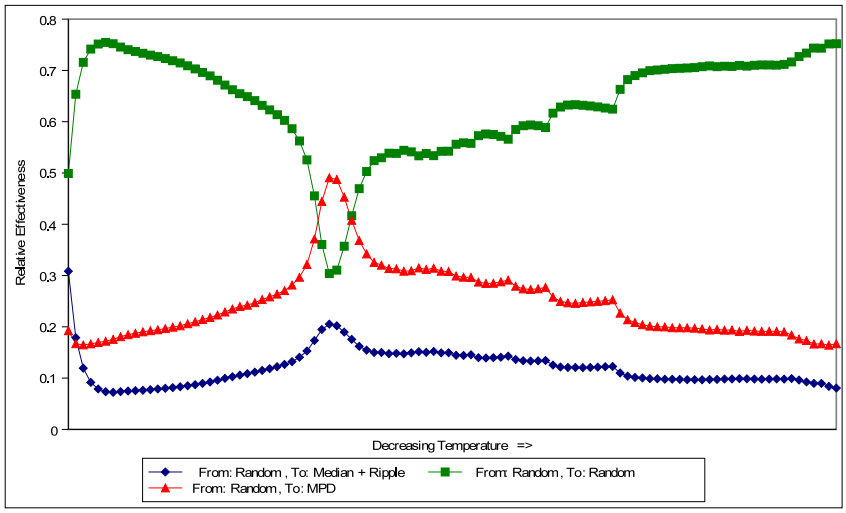


Figure 3.5: Illustration of the probabilities of selecting median placement, MPD, and random moves over the course of an anneal, as the temperature is decreased.

3.4 Experimental Results

Directed moves have the potential to produce high-quality results but can require more run-time than simple moves. In this section, numerical results are presented to support the claim that directed moves (when interspersed with simple moves) can consistently produce *better-quality* placements *for the same amount of work* than if simple moves had been used alone.

For comparison purposes, `KPF` was used to produce placements and `VPR` was used for routing. `KPF` employs criticality history costs [138] and path counting [77] to improve timing-driven results, and can optionally perform BLE-level operations akin to [31]. With these enhancements, `KPF` produces wire length, critical path delay, and run-time results which are on-par with leading academic placers (cf. [31, 138]). It has been empirically validated that `KPF`'s solution quality and run-time are comparable to `VPR` and that it is of sufficiently-high quality that the conclusions drawn in this work regarding directed moves are valid.

The 20 largest designs from the MCNC testsuite [134] were employed for testing. Several architectures were examined—specifically, 1 BLE/CLB, 4 BLE/CLB, and 8 BLE/CLB architectures. Low-stress routing [19] was employed in all cases as this work aimed to quantify the benefits of different types of annealing perturbations on the quality of placement results and not to measure the minimum architectural channel widths. (It is noted that low-stress routing architectures also exist commercially; cf. [1].) Each circuit from the testsuite was run with 5 different seeds with the average over the 5 seeds used for each design.

The quality of `KPF` with directed moves was compared to `KPF` with simple moves alone.

The probabilities of selecting a given directed move (versus a simple move) were determined as described in Section 3.3.4. To produce the quality versus run-time curves, up to 16 different combinations of starting temperatures and moves-per-temperature (`inner_num`) were tested. The average of 5 seeds for each of the 20 benchmark circuits (for the various combinations of directed moves) was computed.

To aid in visualizing the quality-of-results (QOR) trends, the graphs in the following sections present the normalized *geometric means* of the QOR for *the entire suite* (not for individual circuits), plotted against the normalized run-times for the suite. The QOR and run-time values were normalized against the longest-running, simple-move anneal—thus, a value of "1.0" on the y- or x-axis roughly corresponds to the QOR or run-time achievable by an anneal with an `inner_num` of 10 and an initial starting temperature of 20 times the measured standard deviation.[5]

Care was taken to ensure that the annealing parameters were properly tuned for each of the run-time comparison points presented in this work. It is noted that the baseline's nominal (1.0x) run-times may, subjectively, be considered large by industrial standards. Consequently, it is worthwhile to consider the dominance of directed moves across a range of run-times. It is expected that the improvement offered by directed moves at the nominal run-time will typically be smaller than the improvement offered at faster run-times based on the justification presented in Section 3.2.

3.4.1 Commentary on Successful Moves

Very few of the directed moves that were implemented produced meaningful improvement in terms of quality versus run-time compared to using simple moves alone. Moves which did not show early promise were quickly eliminated from further testing. Some moves yielded good improvement but the run-time costs usually outweighed the benefits—more often than not, it would have been better to anneal longer using simple moves than to have employed certain directed moves.

Given the large number of combinations of moves and ways of retaining legality, benchmark sweeps were conducted on 1 BLE/CLB high-utilization architectures to prune combinations of moves which did not appear to be worth pursuing. The success of the moves was qualitatively weighed based on run-time and quality relative to the other move combinations, as well as the effectiveness of the move throughout the anneal. It is worth noting that, when plotted, successful moves produced an effectiveness pattern similar to Figure 3.5, whereas unsuccessful moves approached near-zero effectiveness after a couple of temperature passes.

Examples of move combinations which were attempted and deemed to be unsuccessful are

5 In other words, each trend-line in each graph in the following sections was produced from 1600 individual placements—20 designs times 5 seeds per design times 16 different combinations of starting temperature and moves-per-temperature.

summarized in Table 3.2. Note that this table excludes the results produced by the weighted median and MPD moves because these were among the most successful combinations, and are treated in more detail in the subsequent sections. For each row in this table, the numerical results were produced for the specified directed move in combination with a traditional, simple move. The table highlights the ratios of quality and run-time relative to an anneal without directed moves.

The least successful moves fell into one of two categories:

1. moves that yielded improvement, but were otherwise too computationally expensive; and,

2. moves which did not yield improvement.

The `Domino` move offered some improvement but its run-time excluded it from further consideration. The timing-weighted median placement move offered improvement in wire length but not in timing, but because of the run-time overhead, it was not adequately compelling to use versus a non-weighted median move. The priority list and weighted centroid strategies, on the other hand, tended to produce both worse results and worse run-time. It may be reasonable to place cells at centroids to minimize wire length, but it is not necessarily the case that the centroid is the right place to put a cell to minimize critical path delay. Similarly, swapping critical cells with those that are non-critical prevents the annealer from exploring potentially empty grid sites and does not guarantee a progression to better solutions. It is also worth noting that an improvement in *global* wire length does not necessarily guarantee an improvement in critical path delay. This can arise because the wire length improvement may be made in paths which are non-critical (and, therefore, have no effect on the timing objective). Additionally, to achieve better global wire length, the *localized* lengths of specific driver-sink connections on the critical path may be lengthened by wire length-centric techniques, which can, in turn, worsen circuit timing.

By and large, the most successful directed moves were those which coupled a sufficient amount of randomness with a quick, easy-to-compute, high-quality placement heuristic. The three best examples, from this work, are based on a *random* source selection and, for target selection, either the use of *median placement*, *median placement with cell rippling*, or the minimization of *monotone path deviation*. It is this set of moves which form the basis of the analysis for the remainder of the work.

It is noted that the move selection algorithm (cf. Section 3.3.4) decreases the probabilities of ineffective moves until they approach near-zero values. Consequently, all moves could be turned "on" initially and the probabilities of selecting ineffective moves will be reduced automatically. However, this requires several temperature passes which can slow down the placement. As a result, in the remainder of the work, only the effective moves were enabled during placement.

Table 3.2: Summary of the results for unsuccessful directed moves relative to a baseline anneal.

Source Strategy	Target Strategy	Legality	`timing_tradeoff`	WL Ratio	CP Ratio	CPU Ratio
`Domino`	Graph Colouring	Assign	0	0.97	0.98	4.7×
`Domino`	Graph Colouring	Assign	0.5	0.97	0.99	2.9×
Priority List	Median	Ripple	0	1.00	1.01	1.9×
Priority List	Median	Ripple	0.5	0.97	1.03	1.6×
Priority List	MPD	Ripple	0.5	1.05	1.05	1.5×
Random	Centroid	Assign	0.5	1.03	1.06	1.6×
Random	Priority List	Assign	0.5	1.04	1.06	1.4×
Random	Weighted Median	Assign	0.5	0.89	1.05	1.5×
Random	Weighted Median	Ripple	0.5	0.90	1.04	2.1×

3.4.2 Non-Clustered Architectures

The 1 BLE/CLB architecture was used for investigating directed moves under a variety of scenarios.

3.4.2.1 Device Utilization Tests

In the first experiment, combinations of directed moves were tested on devices with both high-utilization ($\approx 100\%$ utilized) and medium-utilization ($\approx 60\%$ utilized) architectures with the `VPR`-equivalent cost function parameter `timing_tradeoff` $= 0.5$. While the literature has traditionally presented placement results in the context of high-utilization architectures, medium-utilization devices were also considered in this chapter since these tend to be more representative of what is seen in industry.

Results are presented for a set of the more successful directed moves including median placement ("median"), median placement followed by cell rippling ("median + rippling"), weighted median placement (to account for timing criticalities), and monotone path deviation (MPD). Note that, in these figures, as long as the data points for directed moves fall below the baseline's trend-line, the directed moves offer *improvement* in QOR.

The success of the directed moves on a quality versus run-time basis for high-utilization

architectures is presented in Figure 3.6. This diagram shows the results from having used different algorithms for source ("from") cell selection and target ("to") location determination. Infeasibilities are resolved by swapping or rippling, as indicated in the diagram. In these architectures, the use of median placement with rippling resulted in an improvement of 5% in wire length (on average) at the nominal run-time, and at the cost of 2% worsening (on average) in critical path delay. The use of occasional MPD moves resulted in 3% improvement in critical path delay (on average) at the cost of 2% in wire length (on average) at the nominal run-time. Note that this diagram includes information for other, non-successful moves for illustrative purposes; these unsuccessful moves are not considered in the remainder of this work.

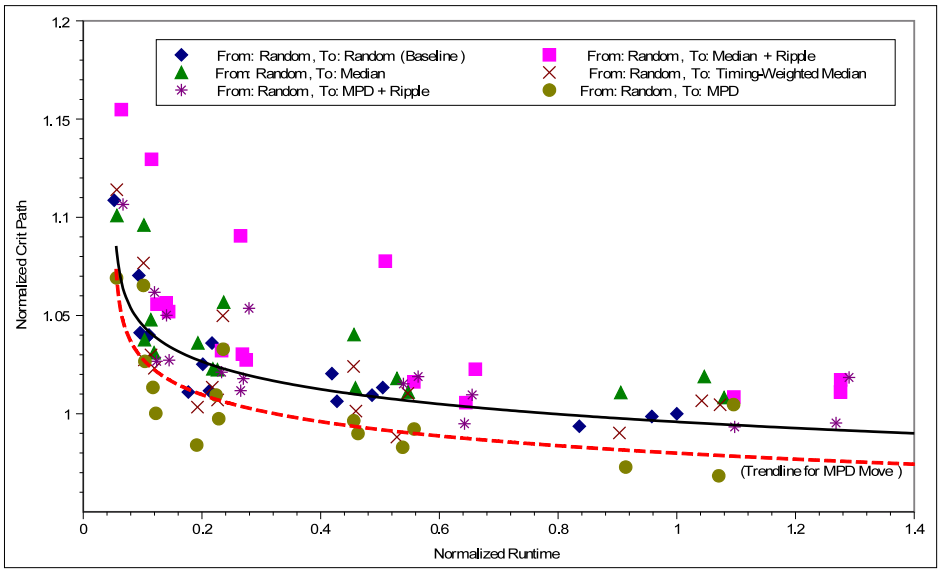
The success of directed moves is much more apparent in medium-utilization architectures, as plotted in Figure 3.7. These curves show that, for the same amount of run-time, median plus rippling moves yielded 9% better wire length (on average) and 2% better critical path delay (on average) at the nominal run-time. The use MPD moves yielded 4% better critical path delay (on average) and 2% better wire length (on average), at the nominal run-time.

It is believed that the greater success of directed moves in the medium-utilization architectures stems from two sources. First, the larger the device, the larger the "search space" of available locations that can be considered by the annealer. The use of directed moves allows this search space to be pruned and focuses the annealer on regions most likely to yield improvement. Second, directed moves are not constrained by shrinking windows, so they can move ill-placed cells further distances at cooler temperatures than random moves. It is important to stress that the improvements achieved by directed moves in the larger architectures are *not* a result of decreased routing congestion as large channel widths were purposefully used to ensure low-stress routing.

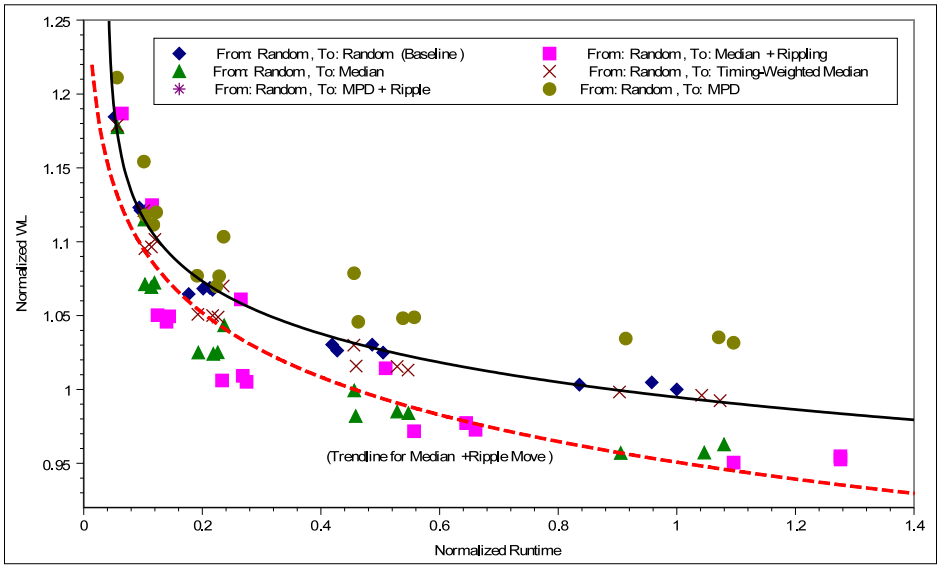
In general, cell rippling was found to improve wire length quality when coupled with the median placement moves, but it typically resulted in a 1% worsening of critical path delay. Similarly, cell rippling was found to reduce the negative impact to wire length in the MPD moves, but the critical path delay improvement was not as good. This is attributable to the fact that the rippling occasionally perturbs timing-critical cells. Due to its success in wire length minimization, the effect of cell rippling was examined primarily with the median placement move in the remainder of this work.

3.4.2.2 Comparison of Timing Trade-offs

Given the improvements to wire length and critical path delay, one might be inclined to conclude that, just by changing the `timing_tradeoff` parameter, one could achieve similar results with simple moves alone. This is **not** the case. The success of directed moves was found to be orthogonal to the design of the cost function—no `VPR`-like parameters can be changed to achieve the same improvement offered by directed moves.

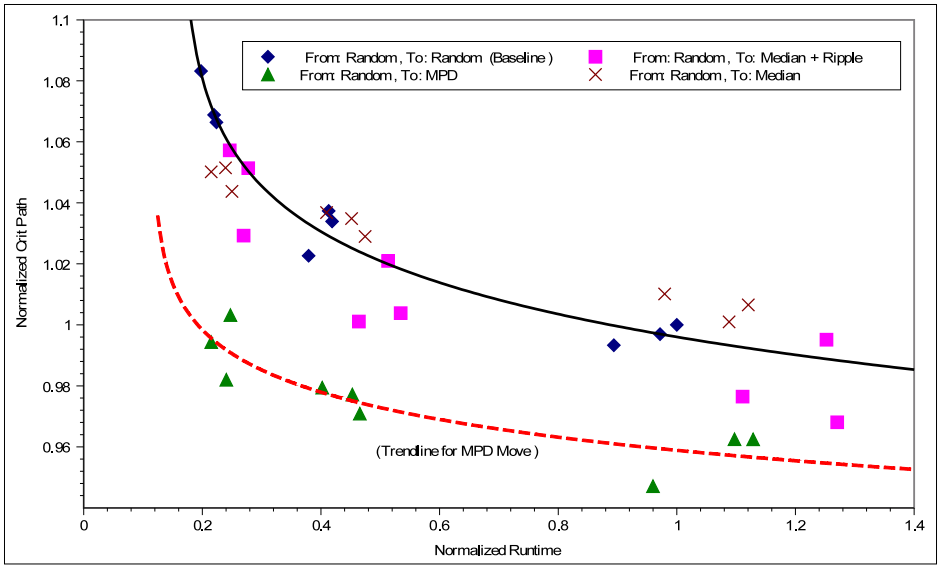


(a)

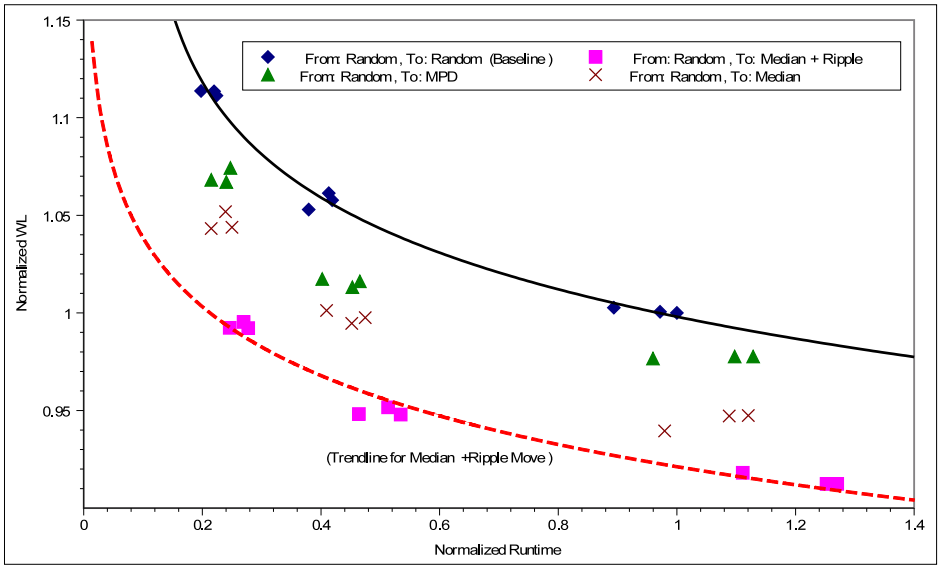


(b)

Figure 3.6: The (a) critical path and (b) wire length quality curves for 1 BLE/CLB high-utilization architectures. This diagram also includes results for some of the non-successful moves for comparative purposes.



(a)



(b)

Figure 3.7: The (a) critical path and (b) wire length quality curves for 1 BLE/CLB medium-utilization architectures.

To validate this assertion, the timing_tradeoff parameter was swept (from 0.1 to 0.9) for placements generated by both simple moves and simple moves with directed moves interspersed. The purpose of this test was to confirm that, for the same amount of run-time, directed moves consistently dominated in both wire length and critical path delays versus simple moves alone. For this test, the median (with rippling) move, the MPD move, and random moves were employed at the same time.

In all cases, directed moves were found to dominate in terms of both critical path and wire length. Figures 3.8 and 3.9 provide two examples of the domination offered by directed moves at different ends of the timing_tradeoff spectrum. The use of directed moves achieved, on average, better results over the entire benchmark suite in terms of wire length and critical path delays, irrespective of the timing_tradeoff parameter in the cost function.

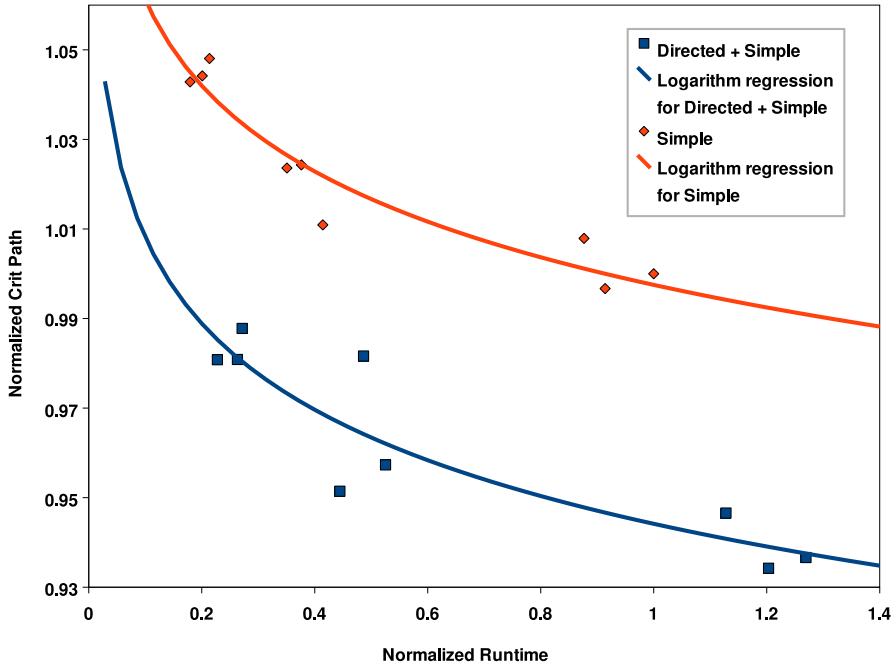
The values of timing_tradeoff were also swept (from 0.1 to 0.9) to determine the best overall quality versus run-time trade-off. The normalized results, acquired on a 1 BLE/CLB, medium-utilization architecture (and computed with respect to the baseline annealer using timing_tradeoff $= 0.5$, inner_num $= 5$) are shown in Figure 3.10. In general, the best balance between wire length and critical path delay was offered by timing_tradeoff $= 0.5$. Using this value (shown by the trend-line), the combination of directed moves achieved a 2% improvement in critical path delay and a 5% improvement in wire length, at the baseline algorithm's nominal (1.0x) run-time.

3.4.2.3 Statistical Variance Measures

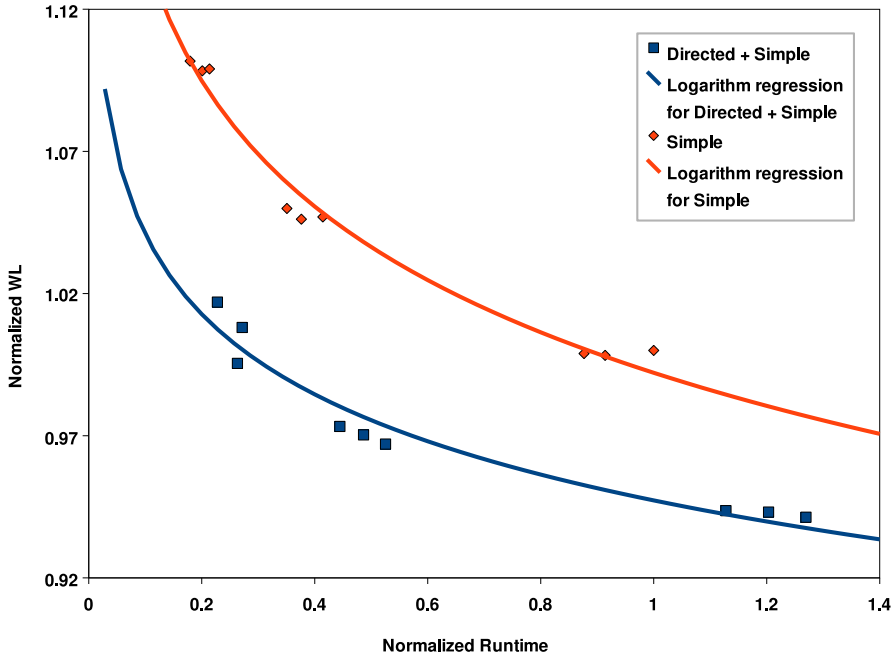
To measure statistical variability, each design in the suite was placed using 39 different seeds, once with simple moves and again with both simple and directed moves enabled (using a mixture of simple, median placement with rippling, and MPD minimizing moves). The averages of the variance and standard deviation of the timing costs and bounding box costs are presented in Table 3.3. In general, directed moves decreased the variance of the placements and tended to make results more repeatable.

Table 3.3: Comparison of average statistical variability in wire length and critical path over the benchmark suite.

	Simple (Baseline)		**Simple & Directed**		**Ratio (vs Baseline)**	
	WL	**CP**	**WL**	**CP**	**WL**	**CP**
Variance	1.34×10^5	1.16×10^{-13}	1.31×10^5	9.91×10^{-14}	0.98	0.85
Std Dev	3.29×10^2	3.08×10^{-7}	3.17×10^2	2.85×10^{-7}	0.96	0.92

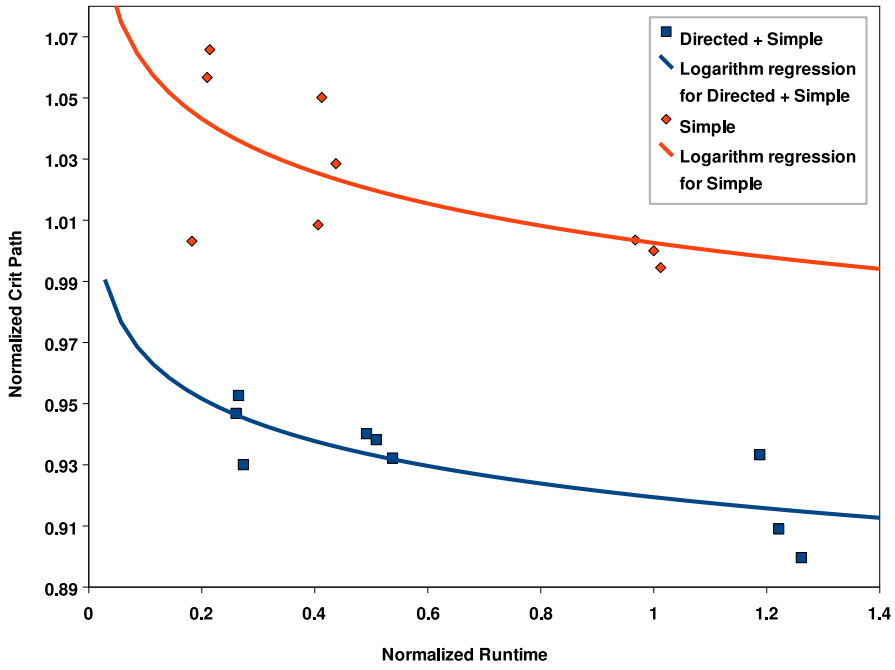


(a)

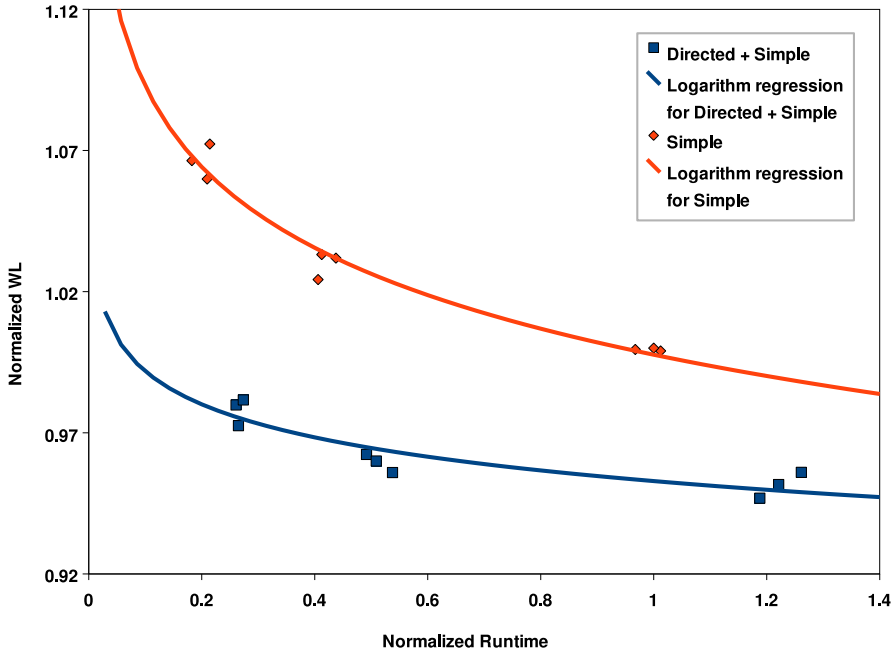


(b)

Figure 3.8: The (a) critical path and (b) wire length quality curves for `timing_tradeoff` $= 0.1$, illustrating the dominance of directed moves versus simple moves alone.

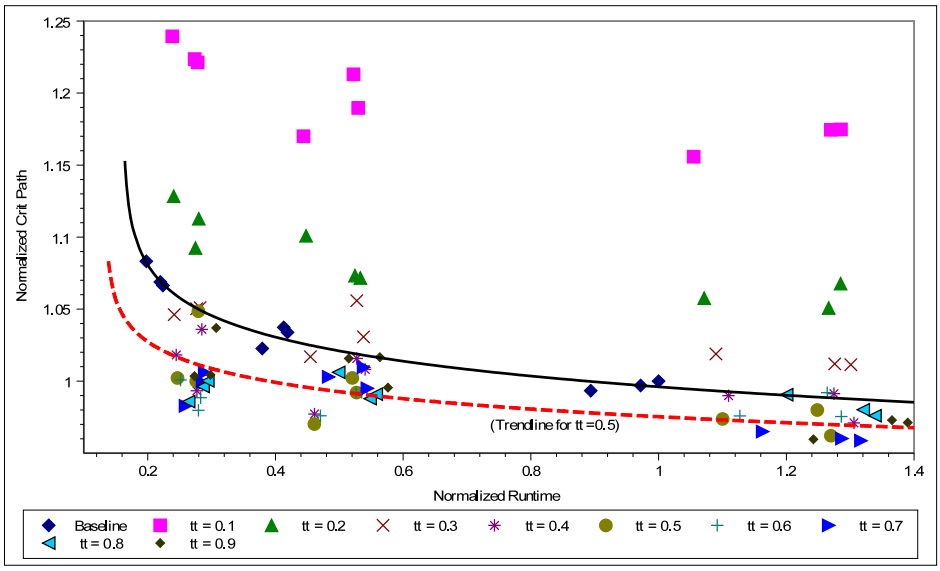


(a)

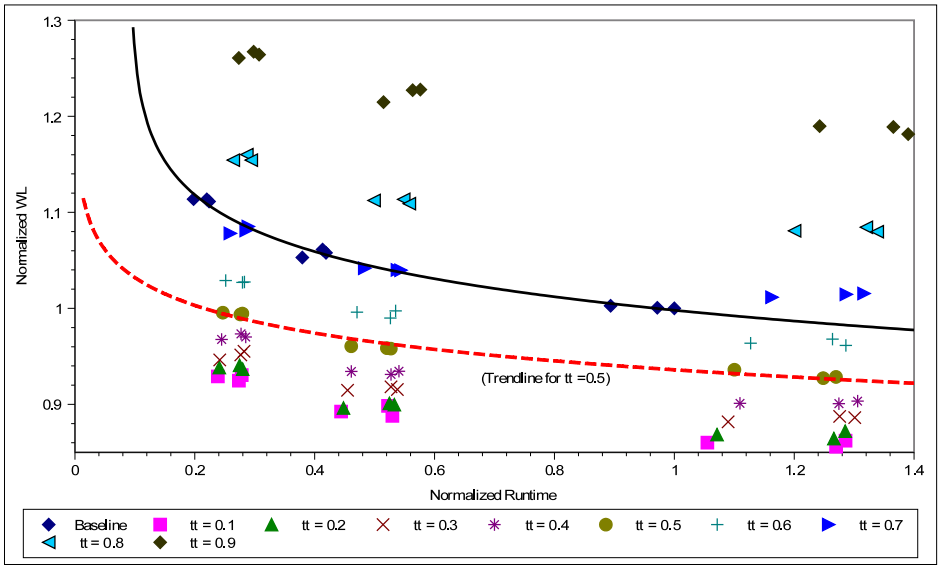


(b)

Figure 3.9: The (a) critical path and (b) wire length quality curves for `timing_tradeoff` $= 0.9$, illustrating the dominance of directed moves versus simple moves alone.



(a)



(b)

Figure 3.10: The (a) critical path and (b) wire length quality curves for various `timing_tradeoff` parameters on a 1 BLE/CLB, medium-utilization architecture. The red-dotted trend-line is shown for the typical `timing_tradeoff` of 0.5, while the blue trend-line is shown for the baseline anneal (with a `timing_tradeoff` of 0.5) without directed moves.

3.4.3 Clustered Architectures

Directed moves have also been tested on architectures with more than 1 BLE per CLB. For these tests, the directed moves were chosen as a combination of simple, median placement with rippling, and MPD minimizing moves, as these had shown the greatest promise in earlier testing.

3.4.3.1 CLB-Level Moves

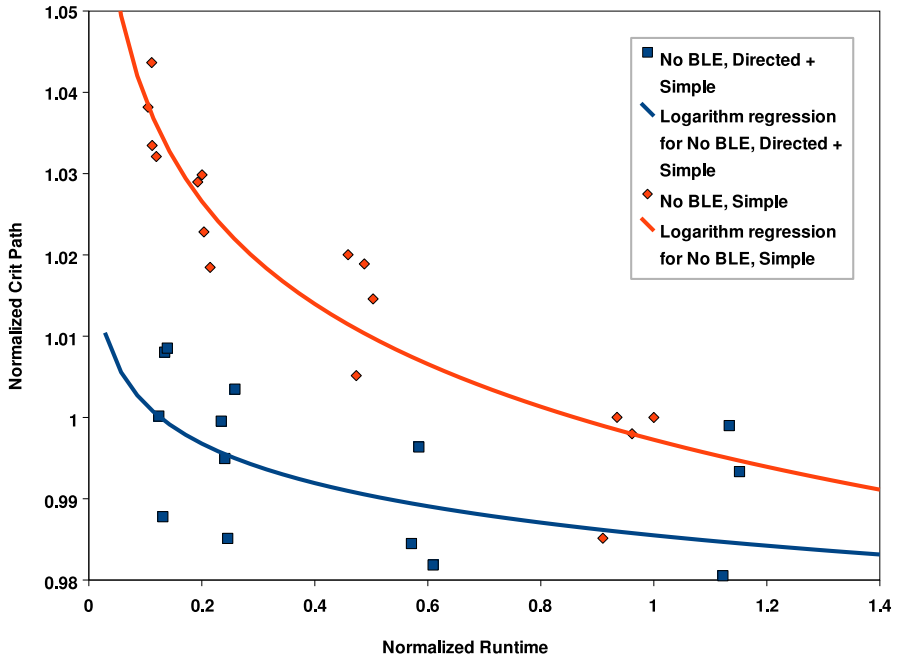
Directed moves were first tested on clustered designs (that is, using CLB-level netlists) without moving BLEs. Figures 3.11 and 3.12 show that directed moves work well on CLB-level netlists on medium-utilization architectures. This is a straightforward application of the technique from 1 BLE/CLB architectures, and the dominance trends are similar. The results for high-utilization architectures are summarized in Table 3.4. (The amount of improvement quoted in these tables represents an approximate reading of the differences in the trend-lines of the dominance charts at the stated run-time points.) Overall, the quality improvement offered by CLB-oriented directed moves for the high-utilization clustered circuits was small. This is due to the small size of the clustered designs and the fact that the baseline annealer does a good job of exploring the search space for such small circuits, so there is less opportunity for improvement.

3.4.3.2 BLE-Level Moves

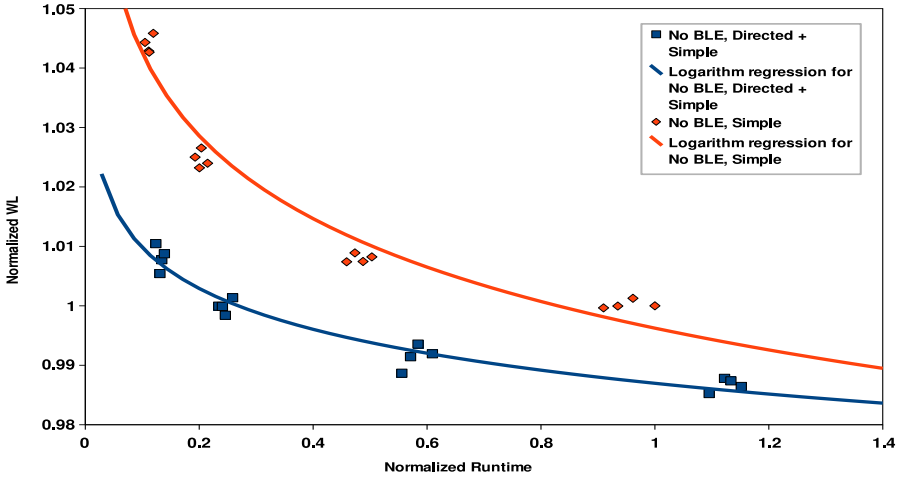
In testing clustered architectures, BLE moves were required to achieve the quality expected from a modern placer [31]; such quality could not be achieved using the traditional pack-then-place flow from `VPR`. BLE moves are conceptually similar to the CLB-level operations performed by annealers like `VPR` except that they place BLEs instead of CLBs. One consequence of performing BLE-level operations is that additional design-rule constraint (DRC) checking and bookkeeping are required; these can be run-time intensive. Thus, to maintain reasonable run-times, fewer BLE moves are typically performed than CLB moves. At the same time, the acceptance rate of BLE moves tends to be smaller because they may not only be rejected by the change in cost function, but also by the DRC checking.

In `KPF`, the BLE moves were implemented after the anneal in a greedy, zero-temperature pass using move effectiveness as the termination criterion. While this differs from the implementation in `SCPlace`, it was the intent of this work to establish a reasonable baseline against which the success of directed moves could be measured and not to address the question of where to perform BLE-level operations.

To justify the use of a zero-temperature BLE-level phase, tests were performed on both 4 and 8 BLE/CLB architectures *without directed moves*. Figures 3.13 and 3.14 summarize the results with and without BLE-level moves for medium-utilization architectures, while the results from

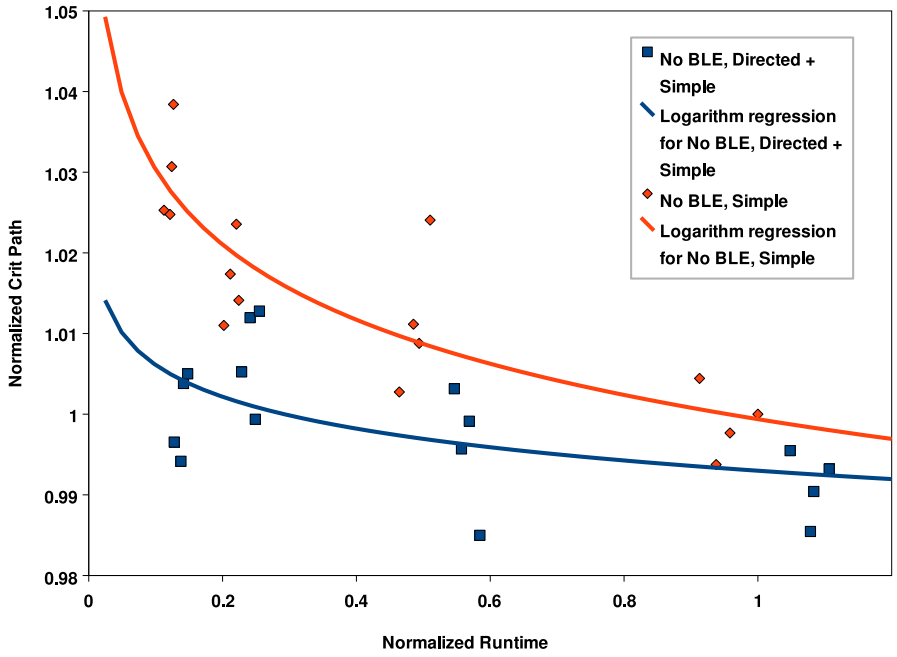


(a)

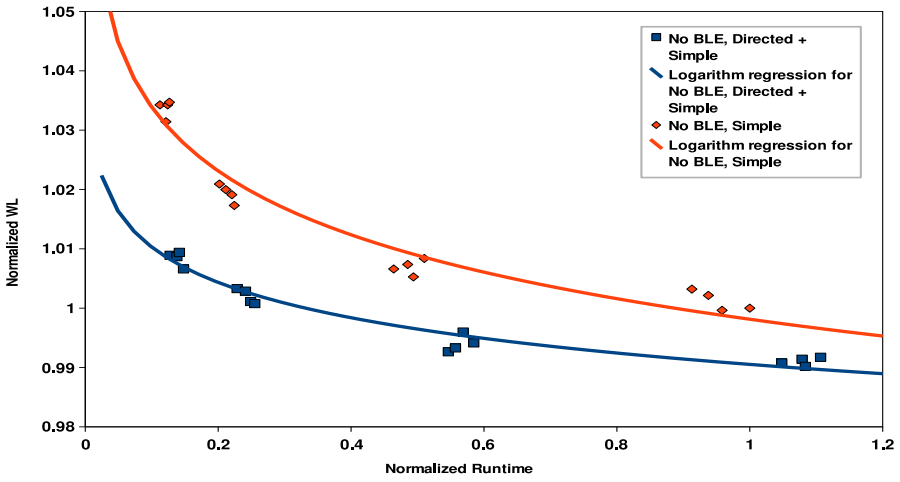


(b)

Figure 3.11: The (a) critical path and (b) wire length quality curves for 4 BLE/CLB medium-utilization architectures with directed moves applied on the CLB-level netlist.



(a)



(b)

Figure 3.12: The (a) critical path and (b) wire length quality curves for 8 BLE/CLB medium-utilization architectures with directed moves applied on the CLB-level netlist.

Table 3.4: Summary of the results for high-utilization clustered architectures with directed moves applied on the CLB-level netlist (negatives indicate improvement).

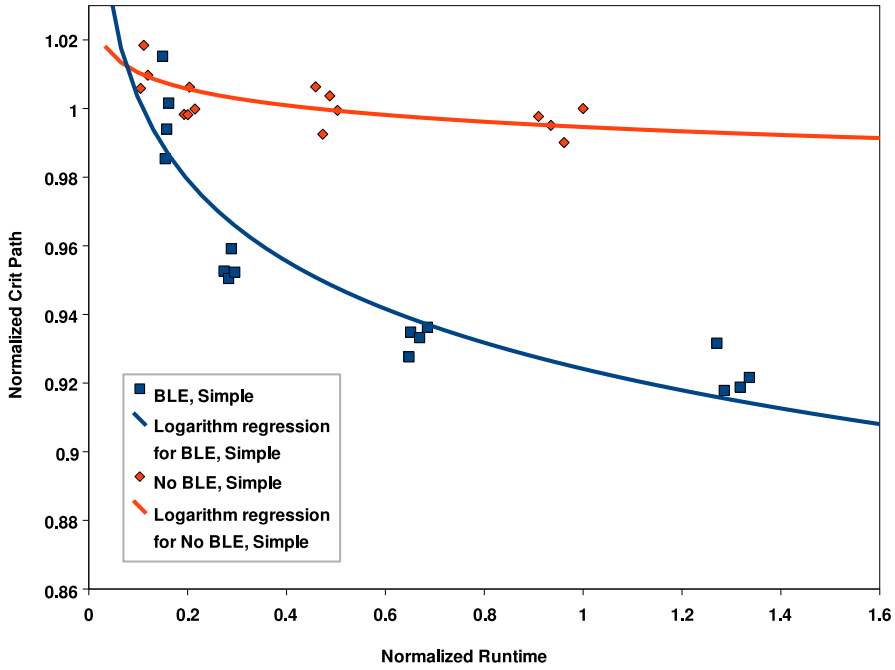
CPU Ratio	Architecture	WL	CP
$\approx 0.25\times$	4 BLE/CLB	-1.5%	-1.5%
$\approx 0.50\times$	4 BLE/CLB	-1%	-1%
$\approx 1.0\times$	4 BLE/CLB	-0.5%	-0.5%
$\approx 0.25\times$	8 BLE/CLB	-1%	-1%
$\approx 0.50\times$	8 BLE/CLB	-0.5%	-0.5%
$\approx 1.0\times$	8 BLE/CLB	$\approx 0\%$	$\approx 0\%$

high-utilization architectures are summarized in Table 3.5. (The amount of improvement quoted in this table represents an approximate reading of the differences in the trend-lines of the dominance charts at the stated run-time points.)

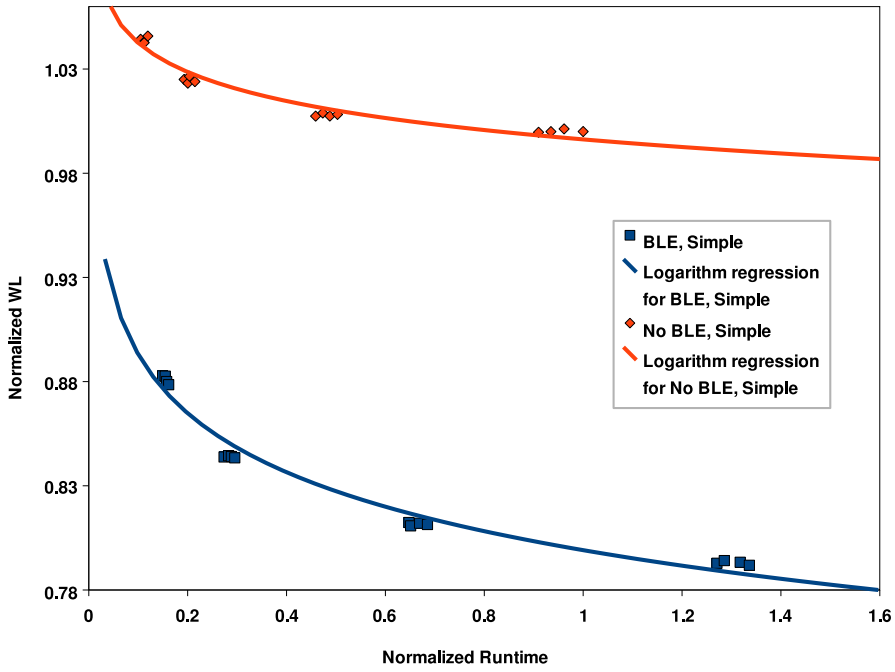
As expected, the wire length and critical path delays improved with more clustered architectures. The overall improvement (in the 8 BLE/CLB medium-utilization case) was a reduction of 25% in wire length and 11% in critical path delay at the nominal run-time, and this is in-line with the amount of improvement reported in [31] for the contribution due to BLE operations during annealing. Hence, it was felt that this implementation offered a reasonable basis for investigating directed moves in clustered architectures.

3.4.3.3 BLE-Level Moves With Directed Moves

The ability for directed moves to work in conjunction with BLE-level moves was also tested. For this test, directed moves were used during both the CLB- and BLE-level phases of the placement and compared to the baseline (where CLB- and BLE-level moves were employed *without* directed moves). The results are shown in Figures 3.15 and 3.16. In general, the directed moves maintained dominance, but the amount of improvement was smaller than without BLE-level moves. Directed moves offered improvement of 4% in wire length and 1% in critical path delay at the 0.2x run-time point, but the improvement at the nominal run-time point was only 2% in wire length and little change in critical path delay. It is believed that this difference may be attributable, in part, to the size of the MCNC benchmarks: in architectures of 4 or 8 BLEs per CLB, the designs are small by modern standards and may not have as much room for QOR improvement once BLE moves are applied, since most of the search space may be explored by the default annealing schedule. The results for high-utilization architectures, summarized in Table 3.6, also exhibited dominance

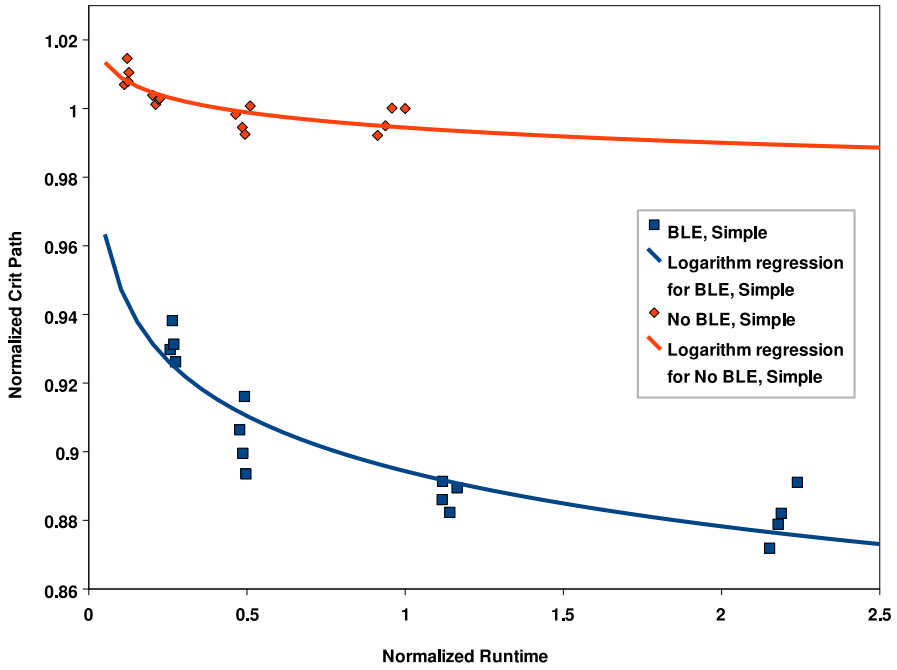


(a)

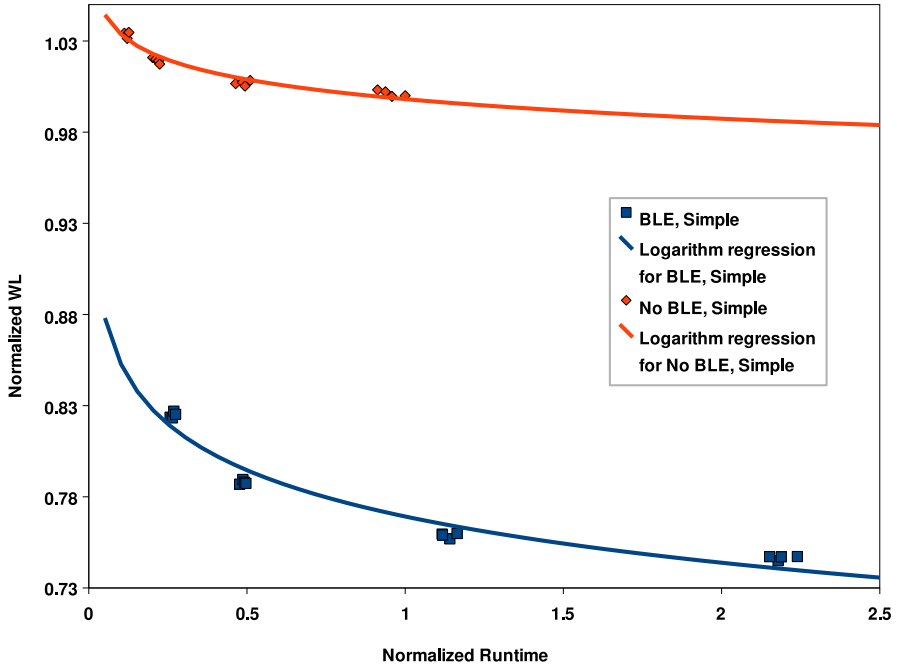


(b)

Figure 3.13: The (a) critical path and (b) wire length quality curves for a 4 BLE/CLB medium-utilization architecture without directed moves but with BLE-level moves.



(a)



(b)

Figure 3.14: The (a) critical path and (b) wire length quality curves for a 8 BLE/CLB medium-utilization architecture without directed moves but with BLE-level moves.

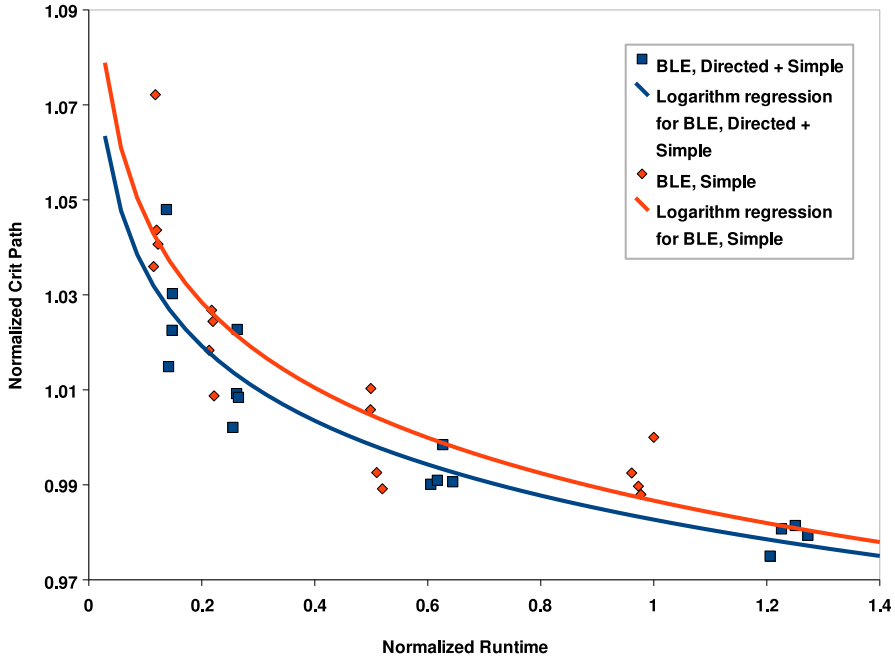
Table 3.5: Summary of the results for high-utilization clustered architectures without directed moves but with BLE-level moves (negatives indicate improvement).

CPU Ratio	Architecture	WL	CP
$\approx 0.25\times$	4 BLE/CLB	-17%	-2%
$\approx 0.75\times$	4 BLE/CLB	-19%	-4%
$\approx 1.0\times$	4 BLE/CLB	-20%	-5%
$\approx 0.25\times$	8 BLE/CLB	-20%	-5%
$\approx 1.0\times$	8 BLE/CLB	-25%	-7%
$\approx 2.0\times$	8 BLE/CLB	-26%	-9%

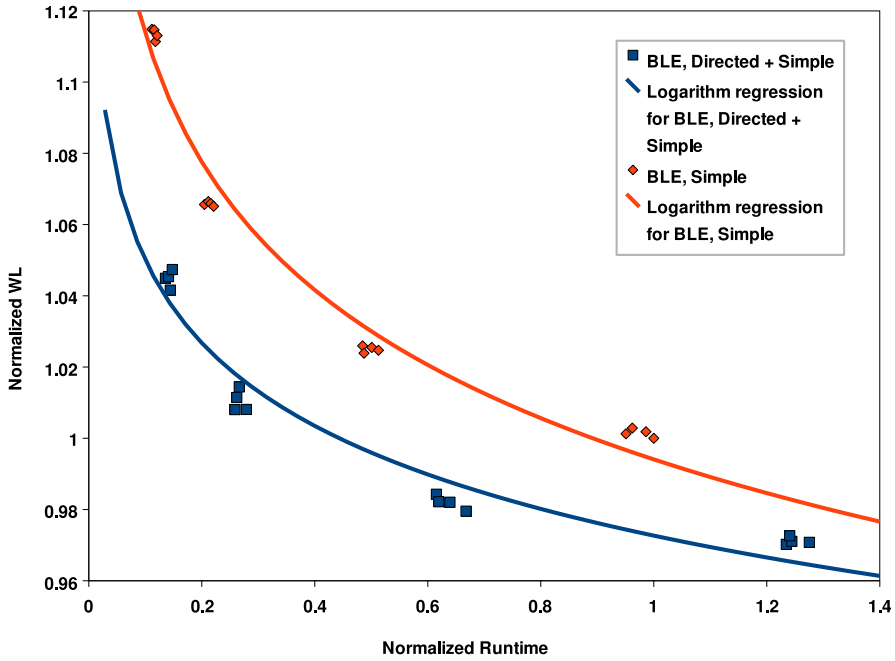
trends, albeit with less overall improvement.

3.5 Conclusion

This chapter described a means of augmenting a traditional, VPR-like FPGA annealer using the concept of directed moves. Multiple types of moves were described, and results were presented that showed that interspersing directed moves into an annealing-based placer led to consistent improvement in QOR (for the same amount of run-time) over an annealer using simple moves alone. Moreover, it was shown that directed moves are useful in (realistic) devices with lower utilization. This work described how directed moves can reduce the statistical variability in placements, which can lead to more repeatable results. Furthermore, it was established that the benefits of directed moves cannot be achieved by changing the annealer's cost function. A new approach for BLE operations was described, and a technique for measuring move effectiveness was also proposed.

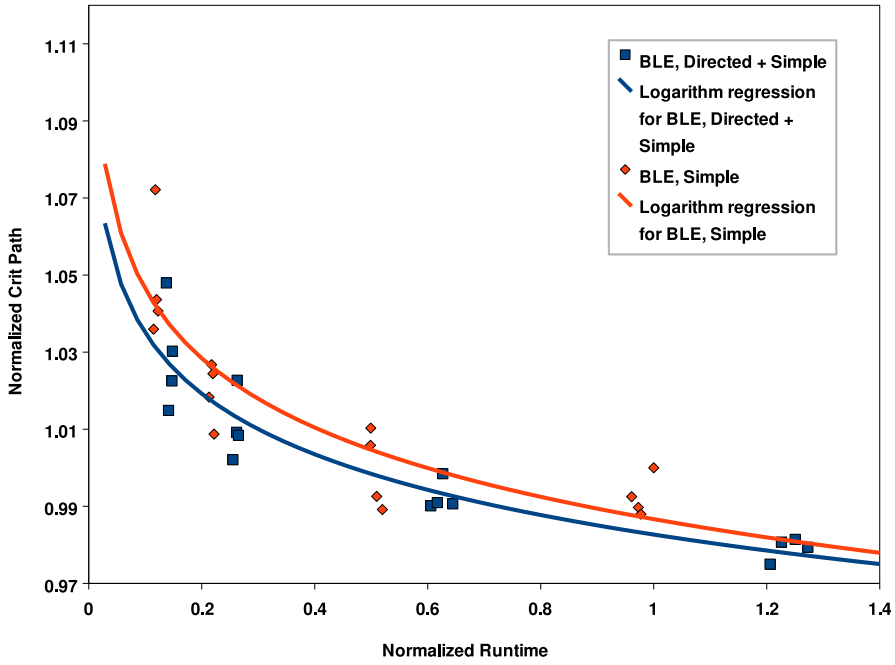


(a)

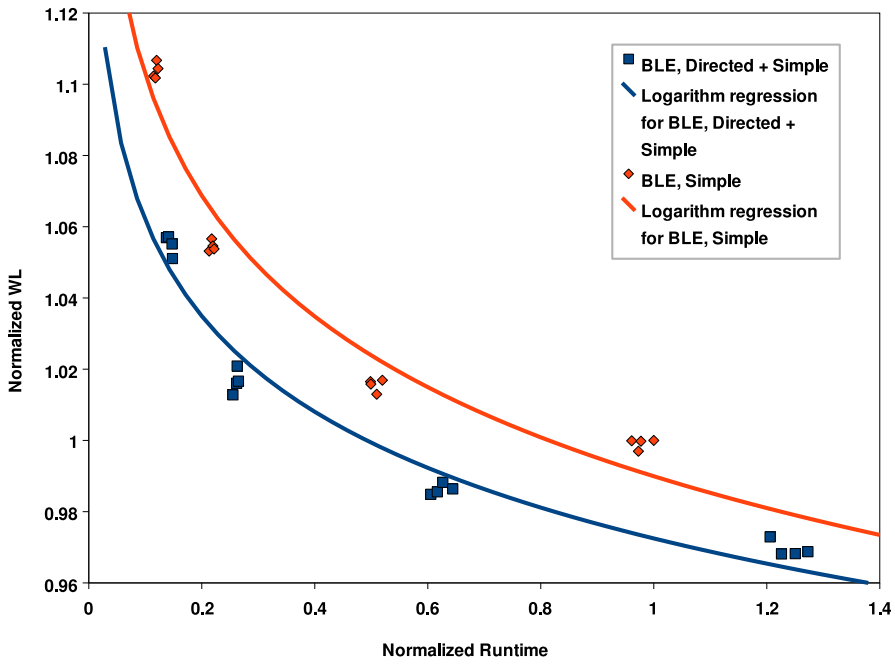


(b)

Figure 3.15: The (a) critical path and (b) wire length quality curves for a 4 BLE/CLB architecture with BLE-level moves and directed moves applied on both the CLB- and BLE-level netlists.



(a)



(b)

Figure 3.16: The (a) critical path and (b) wire length quality curves for a 8 BLE/CLB architecture with BLE-level moves and directed moves applied on both the CLB- and BLE-level netlists.

Table 3.6: Summary of the results for high-utilization clustered architectures with BLE-level moves and directed moves (negatives indicate improvement).

CPU Ratio	Architecture	WL	CP
$\approx 0.25\times$	4 BLE/CLB	-3%	-2%
$\approx 0.5\times$	4 BLE/CLB	-2%	-1%
$\approx 1.0\times$	4 BLE/CLB	-1%	-0.5%
$\approx 0.25\times$	8 BLE/CLB	-2%	-0.5%
$\approx 0.5\times$	8 BLE/CLB	-1%	-0.5%
$\approx 1.0\times$	8 BLE/CLB	-0.5%	$\approx 0\%$

CHAPTER 4

Improving Global Legalization with Directed Moves

4.1 Overview

While directed moves proved to be successful in the context of FPGA annealing-based placement, it was felt that other avenues of VLSI CAD could also benefit. One area, in particular, where it was felt that a stochastic search strategy could be augmented with such moves was in the area of the legalization and floorplan repair of standard cell and mixed-size ASIC designs. [1]

As mentioned in Chapter 2, and as illustrated in Figure 2.5 (d), the legalization problem typically arises during large-scale ASIC placement due to the use of global methods (such as force-directed placement) which produce placements with overlaps. Because of the various objectives that may have been optimized during global placement (such as routing congestion, wire length, and density), the goal of a legalizer should be to perturb the placement as little as possible to preserve the qualities of the global placement.

This chapter demonstrates that a straightforward, top-down approach for legalizing circuits can *reliably* produce feasible placements and floorplans with excellent quality and run-times compared to leading academic tools. The method introduced in this work—named `Whim`—moves only those features which are responsible for violating overlap constraints. `Whim` combines a constraint-based approach [40, 90] with a novel polynomial time, eight-way geometric shifting method based on the concept of *whitespace management* [15, 18]. The quality of this approach is justified across

1 Portions of this chapter were published in [122, 124, 125].

a broad range of benchmarks, and the technique is shown to produce placements which preserve the qualities of the original layouts. Although this approach is described in the context of large-scale VLSI floorplanning problems, it is important to note that `Whim` is equally applicable in other contexts, including rectangle packing [89] and facility layout (e.g., [106]).

The rest of this chapter is organized as follows. The nature of this chapter's top-down approach is described in Section 4.2. Section 4.2.1 presents an approach for legalizing "large" cells, while a technique for legalizing "small" cells is described in Section 4.2.2. Section 4.3 presents numerical results.

4.2 Top-Down Flow for Legalization and Floorplan Repair

A good legalizer possesses several characteristics: (1) robust at resolving overlaps between "large" cells (since "small" cells can be legalized fairly easily *around* them); (2) capable of handling fixed cells; (3) capable of preserving whitespace by shifting cells as little as possible; and (3) good run-time scaling. `Whim` addresses these characteristics by implementing a top-down legalizer whose pseudocode is presented in Figure 4.1. Being top-down, `Whim` divides the placement into manageable "chunks" to improve scalability.

The `Whim` framework works as follows. Initially, all cells are placed into a root partition. Circuit statistics are collected and used to differentiate between the "large" and the "small" cells in this partition. "Large" cells, for example, may consist of all cells greater than 50 times the average cell area in the partition; all others are considered "small".

Large cells are placed via the minimum-movement constraint graph-based floorplanning technique described in Section 4.2.1. Once legalized, the large cells are *fixed* in place. If there are too many "small" cells remaining, the placement region is alternately (and recursively) bisected in the horizontal and vertical directions. Cutlines are located by computing the geometric occupancies of the partitioned region, and placing the cutline so as to balance cell area. Fixed cells which "straddle" the cutline are fixed in both sub-partitions, while small cells are assigned to their closest (under-occupied) partition. Regions are recursively partitioned until the number of small cells in a partition is below a threshold (say, 1000 cells). At this point, the small cell legalizer described in Section 4.2.2 is employed to purge overlaps.

When working back up the partitioning tree, some cells may have been found to be unplaceable in each sub-partition. (Unplaceable cells can arise from incorrect positioning of the cutline or direction of the cut relative to the dimensions of the cells in the sub-partitions.) The two partitions are merged together and a geometric-based technique, described in Section 4.2.2, is employed to place any remaining, unplaced cells. The number of unplaced cells is usually very small, so this operation does not take much time.

```
    Procedure: RECURSIVELEGALIZE
    Inputs: a partition block, B
    Returns: number of unplaced cells in B
1   begin
2     Determine large cells in B via circuit statistics;
3     if the large cells are not already legal then
4       Legalize large cells with large cell floorplanner;
5     fi
6     Fix all of the large cells;
7     if number of small cells ≤ small_cell_threshold then
8       if the small cells are not already legal then
9         Legalize small cells with small cell legalizer;
10      fi
11      Fix all of the successfully-placed small cells;
12    else
13      Determine cut direction;
14      Partition0 ← cells in left (bottom) partition;
15      Partition1 ← cells in right (top) partition;
16      Unplaced0 ← RECURSIVELEGALIZE( Partition0 );
17      Unplaced1 ← RECURSIVELEGALIZE( Partition1 );
18      if Unplaced0 > 0 or Unplaced1 > 0 then
19        Unfix any unplaced cells;
20        Legalize unplaced cells with whitespace manager;
21        Fix any new cells that have been placed;
22      fi
23    fi
24    return number of unplaced cells in B;
25  end
```

Figure 4.1: Outline of `Whim`'s top-down framework.

4.2.1 Legalizing Large Cells

Macrocell legalization schemes have been examined extensively in the context of floorplanning for some time [4, 80, 89]. Annealing-based floorplanners are known to produce good results, but can require large run-times and are generally limited to placing a few hundred cells at once. Alternatively, most published techniques for the geometric placement of macrocells [56, 73, 123] have suffered from large placement perturbations or failure to find legal placements on more complicated problems.

A greedy, search-based floorplanner was implemented which is capable of legalizing $500-800$ cells in reasonable run-time. This floorplanner is used to remove overlap between large cells in each partition of the top-down flow (where "large" is a threshold determined from circuit

statistics in the partition). Operations on a topological constraint graph (TCG) are performed by the floorplanner to achieve legality, and are similar in spirit to [40, 90]. This type of data structure is particularly well-suited to legalization in the presence of fixed obstacles.

The term TCG, in this thesis, refers to the topological constraint graphs originally described in [90]. Such graphs must maintain the "L1 property" [90] which requires that a path exist between every pair of cells, through either direct or transitive arcs, in either the horizontal or vertical directions but not both. Like [90], sequence pairs [89] for the horizontal and vertical directions are first built from the placement by considering cell locations as points, and the TCGs are constructed from the sequence pairs. The sequence pairs are *only* used to maintain the TCGs—the technique presented in this chapter is driven entirely by the slacks on the arcs in the constraint graphs. The sequence pairs merely aid in computing the changes to the graphs rather than manipulating the graphs directly. It is worth noting that the term TCG was coined in [80] in reference to *transitive closure graphs* for floorplanning; on the other hand, this thesis employs the term to refer to constraint graphs which have been *transitively reduced*.

Once the TCGs are constructed, a slack analysis is performed on the TCGs [40, 90, 123] to determine if the placement would fit in both directions. If so, the TCGs are said to admit a "feasible" solution; if not, *directed* "move" and "swap" operations are applied to render the graphs legal. It is important to note that the solution space for manipulating TCGs is very large—$O\left((m!)^2 2^m\right)$ for m blocks [89]—and as such, optimal techniques are impractical for large m.

4.2.1.1 Swap Permutation

A *directed move* heuristic similar to the "swap" operation of Nag and Chaudhary [90] (referred to as an "edge adjustment" in [40]) is employed during TCG legalization. This involves "moving" an edge from a constraint graph in one direction to the constraint graph in the other direction, as shown in Figure 4.2. First, a path-counting heuristic (based on [77]) is applied to the violating constraint graph. This allows the arcs in the graph to be ranked based on their criticality. Next, the longest path in the violating constraint graph is determined, and the critical subgraph is extracted [40]. The reduction edges from this subgraph become the candidates for edge "adjustment" [40]. These candidate arcs are sorted based on their criticality (as computed by the path counting heuristic [77]).

Starting from the most critical candidate arcs, the effects of moving an edge from one constraint graph to the opposite graph are tested until the worst-case slack is improved. Moving an edge between cells A and B in the horizontal constraint graph is as simple as swapping A and B in the horizontal sequence pair (and rebuilding the TCGs)—this establishes a new vertical arc from A to B. (Similarly, swapping in the vertical sequence pair would result in a horizontal arc from B

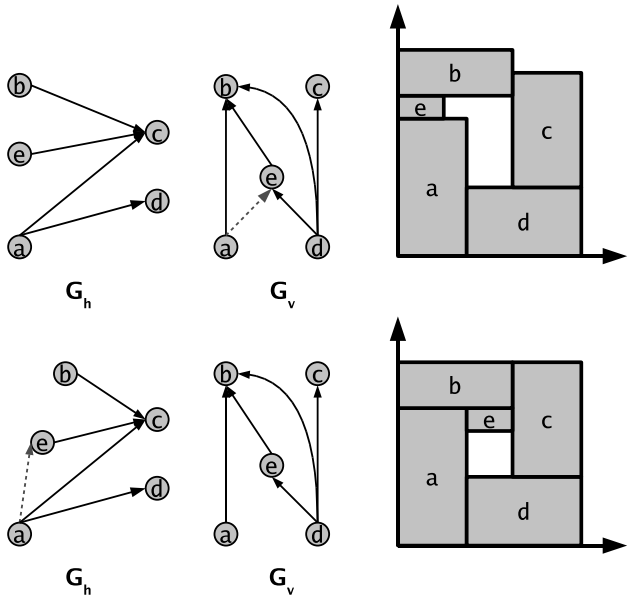


Figure 4.2: A "swap" operation, which moves the edge from $a \rightarrow e$ in G_v to $a \rightarrow e$ in G_h.

to A.) The algorithmic complexity of swapping elements in a sequence pair and rebuilding the constraint graph is the same as moving the arc in the constraint graph in the first place—but, instead of scanning through the constraint graphs to determine edges to add and delete (to preserve the "L1" property), operations on the sequence pair (and the subsequent rebuilding of the constraint graphs) led to a simpler implementation. As in [40] (and unlike the work of [90]), "diverging" and "re-converging" fanouts are accounted for in the critical subgraph—that is, if an edge in the critical path diverges from one node to, say, two nodes (as shown in Figure 4.3), *both* diverging edges will be adjusted at the same time. This extension to [90] allows `Whim` to resolve overlaps more quickly because the floorplanner does not as easily become trapped in local minima.

4.2.1.2 Move Permutation

Although rare, it is possible that a feasible solution may not be found by simply moving edges between constraint graphs. To address this problem, the "move" operation from [90] was implemented with the intent of moving a candidate cell from its current location (in the constraint graph) to another location in either the same (or opposite) constraint graph so as to lessen or remove the criticality of paths passing through the cell. (There is no similar technique in the works of [40,80]. One may view this operation as a "multi-move", because it accomplishes the equivalent of a series of basic operations from [80].)

The "move" operation works as follows. A path counting heuristic is first applied to the

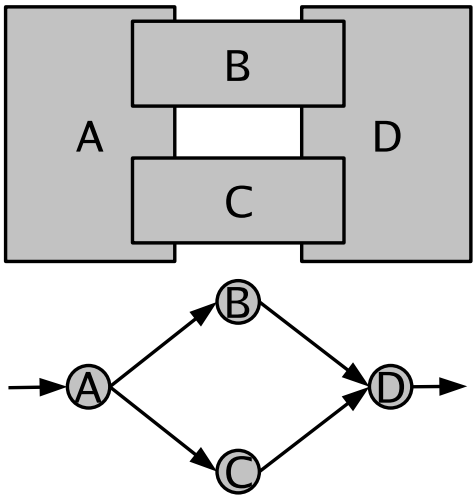


Figure 4.3: Diverging fanouts are shown in a critical horizontal constraint graph formed from overlapping cells. Swapping only one edge (such as $A \rightarrow B$ or $C \rightarrow D$) would not alleviate the criticality in the x-direction—both must be swapped at the same time.

violating constraint graph, and a candidate cell with the largest number of critical paths passing through it is chosen for moving. Candidate arcs (specifically, reduction edges) can be identified in both the horizontal and vertical constraint graphs which possess sufficient slack that the candidate cell can be introduced with no deleterious effects. These candidate arcs are sorted based on their distances from the candidate cell (where the distance is assumed to be the average distance of the two connected cells, prior to floorplanning). The arcs are then tested until a feasible solution is found. The testing is performed as follows: If a cell C is to be inserted between the arc $A \rightarrow B$ in the horizontal graph, cell C needs to be moved between the locations of A and B in the horizontal sequence pair. Cells A and B may not necessarily be adjacent in the sequence pair, so the position of C may be placed randomly between them; each choice gives rise to different constraint graph formulations. The effect of the move is tested, and if it results in a reduction in the critical path, the move is accepted. In general, rewarding moves can be found fairly quickly owing to judicious sorting heuristics.

In practice, this type of *directed* constraint graph adjustment was found to be less efficient than the "swap" operation (described earlier) in terms of the probability of it improving the feasibility of the TCGs. At the same time, the "move" operation plays a key role in reducing overlap when there exists a large number of fixed obstacles in the constraint graph. Movable cells can often become "stuck" between fixed cells in the constraint graph—depending upon constraints in the opposite direction, it may not be possible to eliminate the overlap simply by swapping arcs between graphs. The "move" operation, on the other hand, can relocate movable cells away from these fixed

obstacles and thus eliminate the overlap. In this sense, the directed moves help the search heuristic to escape from local minima and to explore the search-space of more promising operations.

4.2.1.3 Determining the Placement from the Constraint Graphs

Once the constraint graphs are feasible, a linear problem is solved to move cells by as small an amount as possible. The LP formulation follows from [40]. Given TCGs G_h and G_v for the horizontal and vertical directions, module locations x_i and y_i, module heights and widths h_i and w_i, and the height and width of the partitioned regions, W and H, we can solve for the new positions of the cells, x'_i and y'_i (within the partitioned region) using:

$$\min \sum_{i=1}^{n} (\alpha d_{x_i} + \beta d_{y_i}) \tag{4.1}$$

subject to:

$$\begin{gathered}
-d_{x_i} \leq x'_i - x_i \leq d_{x_i} \\
-d_{y_i} \leq y'_i - y_i \leq d_{y_i} \\
x'_j - x'_i \geq \frac{w_i + w_j}{2} \text{ if } \exists e_{ij} \in G_h \\
y'_j - y'_i \geq \frac{h_i + h_j}{2} \text{ if } \exists e_{ij} \in G_v \\
\frac{w_i}{2} \leq x'_i \leq W - \frac{w_i}{2} \\
\frac{h_i}{2} \leq y'_i \leq H - \frac{h_i}{2}
\end{gathered}$$

where α and β are positive values which can be used to preferentially weight individual slacks. (These weights are set to 1 in this work.) `CLP` [37] is used as the LP solver; in practice, the run-time for solving (4.1) is negligible compared to the run-time of the remainder of the legalization flow.

4.2.2 Legalizing Small Cells

As described in Section 4.2.1, large cells are first legalized via a constraint-graph-based floorplanner and then fixed in place. "Small" cells are placed in the remaining gaps of whitespace. The placement of these small cells is accomplished using a novel eight-way shift coupled with a linear program to minimize the amount of shifting.

At the heart of the small cell legalizer is a technique for monitoring gaps of whitespace, termed a "*whitespace manager*". This method was inspired by [15, 18]. In this technique, gaps of whitespace and occupied space are stored as a list of rectangles, and this list is incrementally

updated during legalization. The fundamental operation for monitoring whitespace involves "dividing" the placement region into new, smaller whitespace-rectangles when a cell is inserted into that region.

Consider the placement of a cell within a given (empty) partitioned region. Initially, the whitespace for the empty partition is represented by a single rectangle the size of the partition. When the cell is placed in the partition, the whitespace list is modified by "shredding" the whitespace rectangle and replacing it with adjacent (neighbouring) whitespace rectangles *around* the placed cell. Formally, for each edge e of a cell C that intersects the whitespace rectangle O, and is not collinear with any edge of O, a new whitespace rectangle is created [15]. This new whitespace rectangle is bounded by e and the three other edges of O. (Similarly, the list of occupied space is modified by *adding* a rectangle to represent the placed cell.) In other words, this technique tracks maximally-empty rectangles corresponding to the gaps of whitespace available in the placement, as illustrated in Figure 4.4. By using a 2D interval tree for each set of rectangles [15], the space manager for a gap of whitespace can be queried in $O(\log^2 N + M)$ time, where N is the number of whitespace rectangles in the data structure, and M is the number of rectangles returned by the query. The proof of this complexity is provided in [15].

The whitespace manager affords `Whim` a thorough understanding of where cells may be placed in a non-overlapping fashion. The whitespace manager can be queried to return the nearest gap of whitespace capable of containing a cell of a particular size—as shall be seen, this capability has important uses during floorplan repair, not only in terms of being able to derive overlap-free placements but also in handling row and site alignment constraints.

4.2.2.1 Legalizing Via Eight-Way Shifting

In any given partition during top-down legalization, there may be fixed cells as well as movable cells which need to be legalized. The whitespace management technique (described earlier in

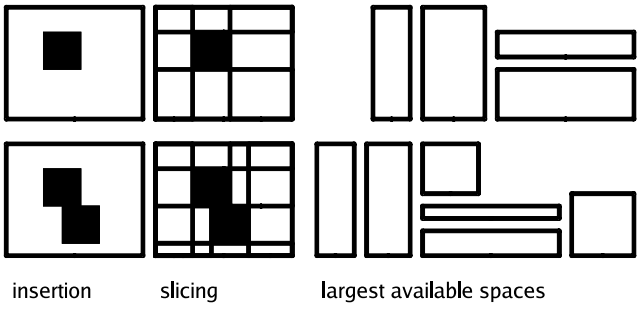


Figure 4.4: Insertion of two cells into the whitespace manager.

this section) can be leveraged to place these cells by considering shifts in all *eight* directions corresponding to packings along the top, bottom, left, right, and in the corners of the partition. The pseudocode for this method is shown in Figure 4.5. The technique works as follows.

Initially, fixed cells within a partition are inserted into the whitespace manager. The list of unplaced cells is sorted based on one of the eight different sorting directions (left-to-right, right-to-left, bottom-to-top, top-to-bottom, and so forth). The list of sorted, unplaced cells is subsequently traversed: for each unplaced cell, the closest gap of whitespace capable of holding that cell is determined. If there exists no such gap, the cell is marked as unplaced, and the count of unplaced cells is increased; otherwise, the cell is inserted into the gap and the whitespace manager is updated accordingly. Once the unplaced cells have been traversed, the best placement found so far is updated. The method repeats for each of the eight sorting directions and the best final solution is returned.

This concept is illustrated in Figure 4.6 for an actual partition. In this diagram, the original layout for a partition is shown in the top-left. The results from 8 different shifts are shown; illegal candidates are marked with "×", and the best (chosen) candidate—the legal solution with the least perturbation—is marked with "✓". Note that a traditional left and/or right shifting (as in [8]) produces two of the candidates marked with "×" (that is, legalization failures). Moreover, calling SMALLLEGALIZE with an already-legal placement results in *no change* to cell positions due to the use of whitespace management, whereas contour-based approaches [8,40,56] typically move cells even if there is no cell overlap.[2]

Once the movable cells are overlap-free, a constraint graph is constructed for the cells in the horizontal direction. This constraint graph is used to represent only the horizontal relationships between cells. Using this graph, an LP similar to (4.1) is solved to determine how to shift cells to minimize their movement from their original positions. Only the x-direction is considered, as this allows standard cells to remain within rows. In effect, *the eight-way shifting technique acts as a directed move for discovering and producing legal constraint graphs, while the LP problem is responsible for the placement of the cells.* The impact on run-time due to this LP-based shifting is negligible.

4.2.2.2 Further Improvements Via Linear Assignment

The eight-way shifting technique may not preserve the original left/right and top/down ordering relationships between small cells. Some cells, for instance, may be assigned to rows (in the

2 As will be described in Section 4.2.3, *two* whitespace managers are actually employed during placement—one for placing macrocells and the other for placing standard cells. This allows standard cells to be aligned to rows and sites.

```
Procedure: SMALLLEGALIZE
Inputs: a partition block, B
Returns: number of unplaced cells in B
1  begin
2    for each fixed cell i in B do
3      Insert cell i into the whitespace manager (as occupied space);
4    od
5    Unplaced ← 0
6    for each of the sorting directions j = 1..8 do
7      Sort the unplaced cells based on j;
8      for each unplaced cell m do
9        Query m's whitespace manager for a free gap;
10       if there are no gaps available then
11         Unplaced ← Unplaced +1;
12       else
13         Insert m into the gap and update whitespace manager;
14       fi
15     od
16     if total movement and Unplaced is best so far then
17       Record placement as the best so far;
18     fi
19     Reset the whitespace manager to its initial state;
20   od
21   Set placement to the best found;
22   Solve LP to minimize cell movement;
23   Use linear assignment to quickly minimize local movement;
24   Fix the placed cells;
25   return number of unplaced cells for the placement;
26 end
```

Figure 4.5: Pseudocode for small cell legalization.

vertical direction) which are further from their original positions than optimal. Typically, the overall movement of cells within a partition can be reduced by $1-3\%$ (even after using the LP to determine a minimum shift in x) via linear assignment. It is important to note that the use of linear assignment is not necessary to achieve legality, but rather can be employed as a method to further minimize the amount of perturbation.

To formulate the assignment problem, candidate locations are positioned at the same spots as the cells in the partition. Cells are connected via arcs (in the assignment problem) only to those locations which are the same size; consequently, the legality of the placement is preserved. The costs on the arcs in the assignment problem represent the distances from a cell's original position to the candidate locations.

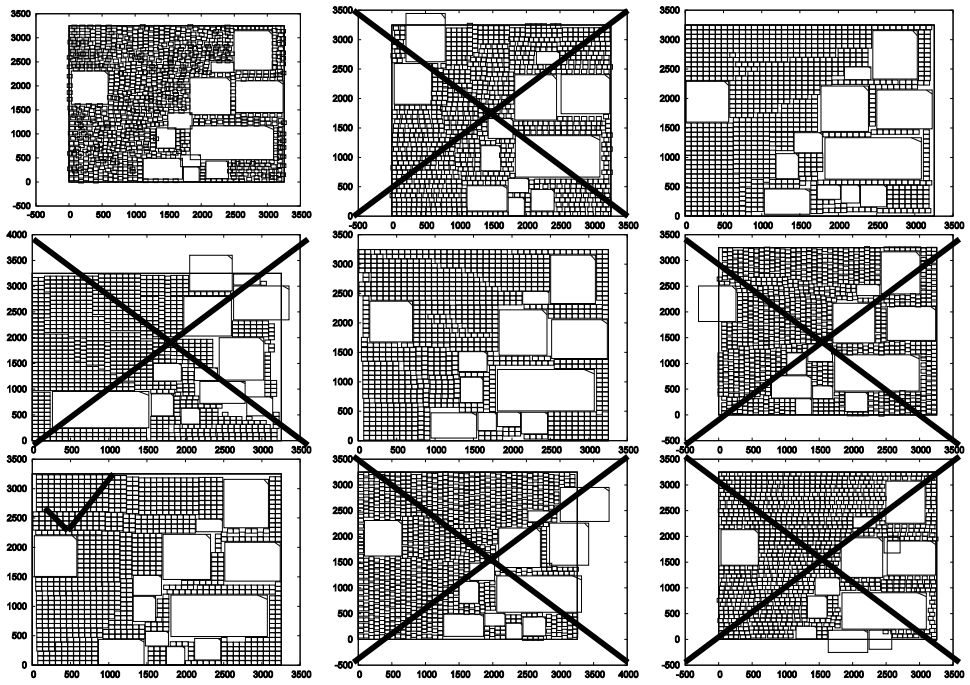


Figure 4.6: An illustration of candidate selection during the 8-way shifting. The original, overlapping placement is shown in the top-left, along with the 8 candidates computed via whitespace management. Only three candidates are legal (non-overlapping); the candidate with the least perturbation from the original is chosen.

The Goldberg-Tarjan push-relabel, minimum-cost flow technique [36,51] is employed to solve the linear assignment. While this method has a worst-case bound of $O(|V|^2|E|\log|V|)$ (for V vertices and E edges), it tends to be very fast in practice because the size of the problem can be artificially limited to achieve good run-times.

4.2.2.3 Commentary On The Success of the Shifting

The combination of whitespace management and eight-way shifting has been found to be *very successful* at finding legal placements. However, owing to the locations of fixed obstacles, the positioning of the cutline, or the size of the partitioned region, the small cell legalization technique may sometimes (though rarely) be unable to place cells within a partition. To address this problem, unplaced cells in a given partition are merged with the unplaced cells in its sibling partitions as the algorithm works its way back up the partitioning tree. These unplaced cells are themselves placed using the whitespace manager. In effect, `Whim` operates in both a top-down and bottom-up fashion, with most cells being placed as the partitions are descended, while unplaced cells are "repaired" as the partitioning works its way up. Fixed cells from the merged partition are inserted into the

whitespace manager. The unplaced cells are sorted by area, from the largest to the smallest blocks. For each unplaced cell, the whitespace manager is queried for the closest gap into which to place the cell. If a gap is found, the cell is placed, the whitespace manager is updated. and the next unplaced cell is selected. In practice, very few designs have been encountered which are not legalizable using this strategy.

4.2.3 Repairing Other Types of Constraints

As mentioned in [86], non-overlap constraints are just one type of constraint considered in legalization and floorplan repair. Other types of constraints, such as region constraints, proximity constraints, and alignment constraints may be of interest. `Whim`'s approach can be extended to repair these constraints during both large- and small-cell legalization. For instance, the TCG floorplanner used for large cell legalization can be augmented to handle such constraints via established approaches [86]. The whitespace manager and eight-way shifting can also be extended to handle such constraints with negligible impact on performance—this is exactly what has been done to ensure row alignments for standard cells.

In the context of standard cell placement, the initial gaps of whitespace for a partition are represented as empty (whitespace) rectangles that correspond to the standard cell rows. This ensures that, when querying the whitespace manager for the nearest available gap of whitespace in which to place a standard cell, the gap which is returned always aligns with a row. In `Whim`, one whitespace manager is used for placing macrocells and one for placing standard cells; additional whitespace managers could be employed for designs with heterogeneous resources (such as RAM and IP blocks) that require placement into (possibly disjoint) discrete placement slots (e.g., [41]).

4.3 Experimental Results

To evaluate `Whim`'s effectiveness, global (overlapping) placements were acquired from two academic ASIC tools—`mPL6` [40] and `NTUplace2` [34]—on the Calypto [94], IBM-HB [94], and ISPD 2005 [92] benchmarks. Since not all placement tools supported soft blocks, the soft blocks were converted to hard blocks with an aspect ratio of 1.0, where appropriate. The characteristics of the designs are presented in Tables 4.1, 4.2, and 4.3.

Three legalizers were run on the output produced by these global placers. The first legalizer was `Floorist` [86] (using the "`-legal -Floorist`" command line switch). The second legalizer was that provided in `mPL6`, which is speculated to be `XDP` [40] (using the "`-mPL_DPonly 1`" switch). The third legalizer was the whitespace-based approach, `Whim`, presented in this work. A timeout of 7200 seconds was imposed on all problems. The total Euclidean movement ("Mvmt"), the half-perimeter wire length ("HPWL"), and CPU run-time ("RT") for the circuits in question are

Table 4.1: Characteristics of the Calypto designs.

Circuit	Cells	Macros	Nets	$\text{Area}_{\text{Largest}}(\%)$	$\frac{\text{Area}_{\text{Largest}}}{\text{Area}_{\text{Smallest}}}$
cal040	1	4605	4607	0.1	650.0
cal098	3200	1212	4673	0.1	529.0
cal336	7	105	147	2.2	1926.0
cal353	217	459	908	7.0	11556.0
cal523	934	1936	4350	0.3	770.0
cal542	7	74	92	20.1	11556.0
cal566	93	1553	5502	1.2	11556.0
cal583	772	1530	3390	0.4	2916.0
cal588	293	495	1111	0.6	900.0
cal643	139	316	598	6.5	6162.0
dct	0	8827	11463	50.0	46332.5

reported in Table 4.5. Overlap and HPWL were determined using `PlaceUtils` [86]. Designs which are not legal are denoted by non-zero overlaps, while designs that crashed or exceeded the run-time threshold are marked accordingly. It is noted that `mPL6` crashed while producing global placements on some IBM-HB designs, and also crashed while legalizing IBM-HB designs. A summary of the geometric means for all legal results is presented in Table 4.4.

In general, `Whim` produced results with significantly less cell movement than the other approaches. Although `Whim` has been presented in the context of floorplan repair, it also works well for standard cell and mixed-size legalization, where `Whim` produced legal placements in all but one case. From log outputs, it is believed that `mPL6` performs compaction and greedy swapping to improve HPWL—in contrast, neither `Floorist` nor `Whim` perform detailed placement to improve or compact designs. Further HPWL improvement could be obtained at the cost of more cell movement by performing detailed improvement (as in [8, 40]). An example of a legalized placement produced by `Whim` is presented in Figure 4.7.

In general, the global placement tools produced reasonably overlap-free global placements on standard, mixed-size benchmarks [2, 92]; yet, many placements were far from legal on those benchmarks with characteristics closer to those of floorplanning problems [94]. Large overlaps have been observed for several global placements produced by `mPL6` and `NTUplace2`. (These tools were run using command line parameters as provided by the authors of the respective tools.) In Figure 4.8, an example of a global placement with significant overlap (i.e., many cells have

Table 4.2: Characteristics of the IBM-HB designs.

Circuit	Cells	Macros	Nets	$\text{Area}_{\text{Largest}}(\%)$	$\frac{\text{Area}_{\text{Largest}}}{\text{Area}_{\text{Smallest}}}$
ibm01	0	911	5829	26.2	50305.1
ibm02	0	1471	8508	45.9	17325.5
ibm03	0	1289	10279	44.1	194393.3
ibm04	0	1584	12456	36.2	74745.8
ibm06	0	749	9963	55.2	104642.3
ibm07	0	1120	15047	18.9	2269.8
ibm08	3	1266	16075	48.3	294180.9
ibm09	1	1112	18913	21.7	172487.7
ibm10	177	1418	27508	19.6	418419.0
ibm11	0	1497	27477	17.9	56636.7
ibm12	284	949	26320	26.1	435131.8
ibm13	40	914	27011	17.1	194393.3
ibm14	58	1577	43062	8.0	105018.9
ibm15	29	1383	52779	44.2	358763.2
ibm16	20	1071	47821	7.7	182911.9
ibm17	160	1282	56517	3.9	74008.5
ibm18	0	943	42200	3.8	18990.6

been placed on top of one-another by the global placement tools) is presented, as well as the legalized result produced by `Whim`. Over all benchmarks, `Whim` produced legal placements in all but one case whereas `Floorist` and `mPL6` produced numerous illegal placements, crashed, or required significant (7200+ seconds) run-time. (Although, for poor global placements such as those produced by `mPL6` on the IBM-HB circuits, the resulting movement and HPWL increase was, understandably, large.) It is felt that this serves as a testament to the robustness of this approach.

4.4 Conclusion

This chapter described the implementation of a top-down strategy for floorplan repair which employs a combination of constraint graphs, linear programming, and a novel geometric shifting technique to remove overlap between cells. Unlike some approaches, this method moves only those features which are responsible for violating overlap constraints, thereby making it a versatile way to post-process the outputs of global floorplanners and placers. The effectiveness of this

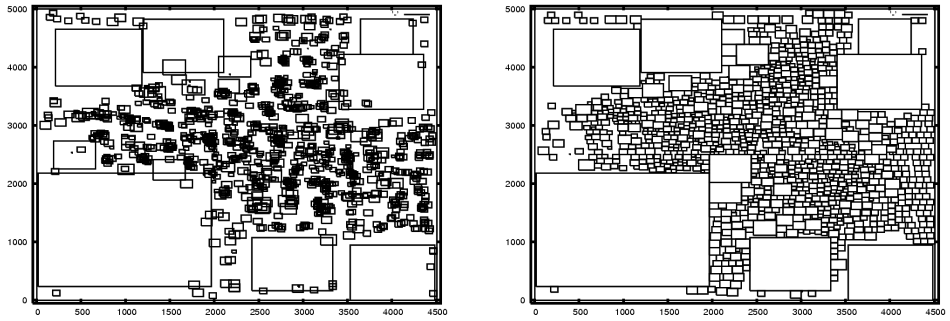


Figure 4.7: A global placement of `ibm09` (from the ICCAD04 suite) and its legalized counterpart produced by `Whim`.

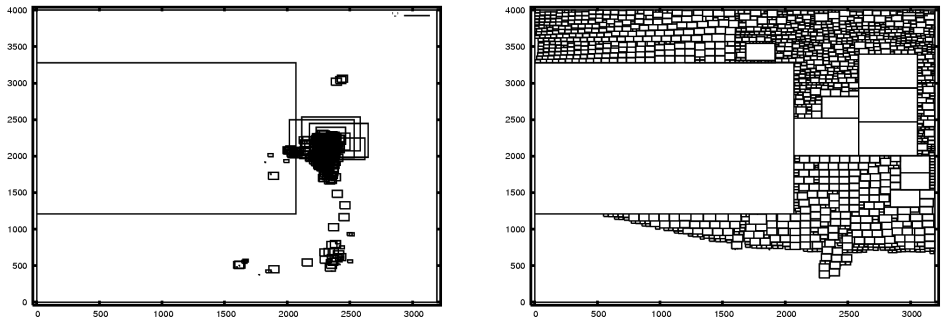


Figure 4.8: `Whim` was found to be remarkably robust in its ability to produce valid, legalized results even when presented with heavily-overlapping global placements.

Table 4.3: Characteristics of the ISPD 2006 suite. Note that these designs do not possess movable macrocells.

	General Statistics		Standard Cell Widths		
Circuit	**Cells**	**Nets**	**Min**	**Max**	**Median**
adaptec1	210904	221142	3	77	13
adaptec2	254457	266009	3	75	8
adaptec3	450927	466758	3	80	10
adaptec4	494716	515951	3	80	10
bigblue1	277604	284479	3	43	11
bigblue2	534782	577235	3	104	9
bigblue3	1093034	1123170	3	104	8
bigblue4	2169183	2229886	3	104	8

Table 4.4: Comparison of `Whim` versus other tools.

	vs. `Floorist`			**vs.** `mPL6`		
Suite	**Mvmt**	**HPWL**	**RT**	**Mvmt**	**HPWL**	**RT**
Calypto	1.02	**0.87**	**0.23**	**0.91**	**0.79**	**0.20**
IBM-HB	**0.95**	**0.96**	**0.05**	n/a	n/a	n/a
ISPD 2005	**0.85**	**0.87**	**0.22**	**0.57**	1.05	**0.21**

algorithm was quantified across a broad range of floorplans produced by multiple tools. `Whim` succeeded in producing valid placements in almost all cases, while requiring only *one fifth* the run-time and producing placements with 4 to 13% less HPWL and up to 43% less movement than leading methods.

Table 4.5: Performance of `Floorist`, `mPL6`, and `Whim` on various benchmark designs.

		Global		Floorist				mPL6				Whim			
Tool	Ckt	OL (%)	HPWL	OL (%)	Mvmt	HPWL	RT (s)	OL (%)	Mvmt	HPWL	RT (s)	OL (%)	Mvmt	HPWL	RT (s)
Calypto															
mPL6	cal040	394.6	3.01e04	*run-time exceeded*			> 7200	0.0	4.12e05	6.63e05	186.5	0.0	2.64e05	2.88e05	8.1
	cal098	118.5	3.03e05	0.0	1.22e05	3.83e05	201.5	0.0	1.31e05	3.30e05	33.6	0.0	1.29e05	3.17e05	15.3
	cal336	1550.5	1.67e03	0.0	4.52e04	5.40e04	0.5	0.0	7.06e04	9.53e04	2.8	0.0	3.30e04	4.07e04	0.8
	cal353	1450.7	9.75e03	0.5	6.34e04	9.84e04	22.9	0.0	8.35e04	1.39e05	24.1	0.0	5.54e04	1.00e05	1.5
	cal523	291.5	6.04e05	0.0	2.59e05	7.18e05	771.8	*crashed*				0.0	2.69e05	5.98e05	12.5
	cal542	567.9	9.38e02	0.0	1.05e04	1.42e04	0.2	0.0	1.60e04	1.79e04	1.1	0.0	5.73e03	1.16e04	0.6
	cal566	243.8	1.06e05	0.7	3.55e05	9.52e05	951.3	0.0	3.88e05	1.85e06	41.6	0.0	2.83e05	9.66e05	7.2
	cal583	427.5	6.52e04	0.1	2.76e05	4.48e05	565.8	0.0	3.19e05	6.21e05	109.3	0.0	1.77e05	3.80e05	12.5
	cal588	258.0	8.14e04	1.3	3.09e04	1.03e05	16.3	0.0	2.36e04	8.53e04	4.6	0.0	2.89e04	7.83e04	1.6
	cal643	394.5	1.10e04	0.2	2.77e04	4.79e04	6.9	0.0	3.75e04	6.12e04	9.9	0.0	2.54e04	5.17e04	0.9
	dct	44.4	1.52e05	*run-time exceeded*			> 7200	*crashed*				0.0	2.54e05	3.28e05	16.1
NTUplace2	cal040	86.1	1.05e05	0.4	2.88e05	4.43e05	4232.4	0.0	4.15e05	7.11e05	718.8	0.2	7.87e04	2.08e05	13.1
	cal098	49599.9	2.75e04	*run-time exceeded*			> 7200	0.0	1.85e06	1.43e06	818.4	0.0	3.04e06	9.98e05	19.9
	cal336	1867.0	4.80e03	0.0	4.32e04	6.69e04	0.4	0.0	7.60e04	8.60e04	1.6	0.0	3.69e04	3.92e04	0.8
	cal353	12791.1	8.99e03	0.0	1.70e05	2.17e05	238.6	0.0	2.06e05	2.72e05	11.4	0.0	1.62e05	1.79e05	1.3
	cal523	37617.5	9.62e04	*run-time exceeded*			> 7200	*crashed*				0.0	2.26e06	2.93e06	11.6
	cal542	42.1	6.28e03	0.0	2.65e03	9.30e03	0.2	0.0	4.25e03	1.13e04	1.5	0.0	4.05e03	1.07e04	0.6
	cal566	70.1	4.16e05	0.0	9.35e04	6.87e05	380.6	*crashed*				0.0	6.25e04	5.97e05	5.2
	cal583	26237.1	8.92e04	*run-time exceeded*			> 7200	0.0	1.81e06	1.43e06	103.5	0.0	1.72e06	1.22e06	7.8
	cal588	14224.5	1.07e04	0.4	1.77e05	2.69e05	286.0	0.0	2.81e05	3.40e05	16.5	0.0	1.55e05	2.46e05	1.4
	cal643	94.6	2.55e04	0.0	1.07e04	3.72e04	3.0	0.0	1.03e04	3.57e04	2.0	0.0	9.40e03	3.50e04	0.9
	dct	16.8	2.10e05	*run-time exceeded*			> 7200	*crashed*				0.0	8.68e04	2.75e05	14.1
IBM-HB															
mPL6	ibm01	2980.6	7.33e05	0.0	7.00e05	7.94e06	1727.6	*crashed*				0.0	5.42e05	6.49e06	2.0
	ibm02	1659.5	1.30e06	*run-time exceeded*			> 7200	*crashed*				0.0	1.80e06	1.44e07	5.1
	ibm03	2074.0	1.39e06	*run-time exceeded*			> 7200	*crashed*				0.0	1.78e06	1.97e07	6.6
	ibm04	4001.5	2.40e06	*run-time exceeded*			> 7200	*crashed*				0.0	1.91e06	2.37e07	3.6
	ibm06	3103.5	8.11e05	2.6	9.45e05	2.22e07	3167.4	*crashed*				0.0	7.58e05	2.38e07	2.0
	ibm07	7036.1	1.89e06	0.0	1.53e06	5.01e07	5325.8	*crashed*				0.0	1.48e06	4.54e07	6.9
	ibm11	2166.5	5.51e06	*run-time exceeded*			> 7200	*crashed*				0.0	2.34e06	6.59e07	4.5
	ibm16	949.9	1.90e07	3.7	1.99e06	1.98e08	938.9	*crashed*				0.0	1.83e06	2.19e08	16.5
	ibm17	2241.7	2.61e07	0.5	1.79e06	2.34e08	1342.3	*crashed*				0.0	1.97e06	2.81e08	10.7
	ibm18	1351.3	1.50e07	0.5	1.29e06	1.18e08	347.8	*crashed*				0.0	1.39e06	1.73e08	7.0
NTUplace2	ibm01	44.2	2.32e06	0.3	2.02e05	4.33e06	320.6	*crashed*				0.0	1.05e05	2.91e06	2.1
	ibm02	28.3	4.96e06	0.1	2.74e05	7.76e06	449.7	*crashed*				0.0	1.79e05	5.90e06	2.4
	ibm03	16.6	7.81e06	0.1	2.90e05	9.21e06	87.5	*crashed*				0.0	1.28e05	8.28e06	2.9
	ibm04	21.3	8.69e06	0.1	4.00e05	1.21e07	479.8	*crashed*				0.0	1.63e05	9.47e06	2.3
	ibm06	28.5	5.93e06	0.1	1.15e05	7.81e06	26.6	*crashed*				0.0	1.44e05	7.49e06	2.3
	ibm07	22.0	1.36e07	0.1	1.49e05	1.52e07	79.1	*crashed*				0.0	1.44e05	1.46e07	3.0
	ibm08	22.6	1.51e07	0.1	1.61e05	1.74e07	94.1	*crashed*				0.0	1.44e05	1.65e07	4.0
	ibm09	30.8	1.30e07	0.0	2.16e05	1.62e07	73.2	*crashed*				0.0	2.05e05	1.63e07	4.3
	ibm10	14.7	3.82e07	0.0	5.84e05	4.35e07	305.9	*crashed*				0.0	3.99e05	4.22e07	8.3
	ibm11	27.4	2.22e07	0.0	3.70e05	3.25e07	466.0	*crashed*				0.0	3.20e05	2.69e07	3.8
	ibm12	14.1	4.79e07	0.0	6.17e05	5.33e07	40.5	*crashed*				0.0	5.30e05	5.13e07	7.1
	ibm13	23.5	2.98e07	0.0	2.80e05	3.41e07	73.8	*crashed*				0.0	3.00e05	3.40e07	8.2
	ibm14	32.9	5.70e07	0.1	3.30e05	6.28e07	498.0	*crashed*				0.0	3.20e05	6.16e07	8.1
	ibm15	21.9	6.78e07	0.0	6.75e05	8.19e07	311.8	*crashed*				0.0	3.97e05	7.86e07	8.9
	ibm16	23.8	8.51e07	0.0	3.51e05	9.53e07	211.2	*crashed*				0.0	4.08e05	9.34e07	7.8
	ibm17	28.4	1.36e08	0.0	3.24e05	1.46e08	155.1	*crashed*				0.0	5.34e05	1.42e08	11.5
	ibm18	37.9	6.63e07	0.1	2.12e05	7.50e07	56.6	*crashed*				0.0	2.28e05	7.46e07	7.1
ISPD 2005															
mPL6	adaptec1	17.6	8.01e07	0.0	8.42e06	8.86e07	2297.3	0.0	1.31e07	7.79e07	883.1	0.0	7.38e06	8.05e07	159.8
	adaptec2	19.8	9.35e07	0.0	2.74e07	1.19e08	4948.4	0.0	2.97e07	9.20e07	883.3	0.0	1.91e07	9.67e07	224.0
	adaptec3	15.4	2.15e08	*run-time exceeded*			> 7200	0.0	7.06e07	2.14e08	1776.5	0.0	4.87e07	2.24e08	414.5
	adaptec4	11.6	1.98e08	*run-time exceeded*			> 7200	0.0	6.95e07	1.94e08	1901.2	0.0	4.41e07	2.04e08	396.0
	bigblue1	12.8	1.02e08	0.0	1.46e07	1.16e08	140.1	0.0	2.05e07	9.68e07	1066.9	0.0	9.09e06	1.00e08	226.9
	bigblue2	13.0	1.57e08	0.0	2.71e07	1.79e08	6962.8	0.0	5.81e07	1.52e08	2932.9	0.0	2.78e07	1.62e08	570.7
	bigblue3	24.2	3.60e08	*run-time exceeded*			> 7200	0.0	1.96e+08	3.44e+08	3970.2	0.0	9.39e+07	3.64e+08	1056.1
	bigblue4	20.0	8.79e08	*run-time exceeded*			> 7200	*run-time exceeded*			> 7200	0.0	1.06e08	9.33e08	1622.3
NTUplace2	adaptec1	23.2	8.69e07	0.0	1.02e07	9.82e07	1893.9	8.9	3.96e07	8.07e07	1624.0	0.0	8.85e06	8.65e07	150.5
	adaptec2	16.8	1.05e08	0.0	1.39e07	1.15e08	272.8	5.2	4.64e07	9.63e07	1807.8	0.0	1.52e07	1.05e08	172.0
	adaptec3	16.6	2.43e08	*run-time exceeded*			> 7200	9.6	3.87e07	2.20e08	2952.5	0.0	6.16e07	2.44e08	329.7
	adaptec4	10.7	2.16e08	0.0	5.69e07	2.81e08	3988.5	5.1	3.79e07	1.97e08	2850.2	0.0	3.38e07	2.15e08	538.6
	bigblue1	13.2	1.11e08	0.0	6.82e06	1.18e08	64.5	0.6	3.44e07	1.05e08	1955.8	0.0	7.28e06	1.10e08	204.8
	bigblue2	14.8	1.69e08	0.0	2.41e07	1.90e08	6267.2	5.6	2.85e07	1.53e08	3488.0	0.0	2.29e07	1.69e08	730.5
	bigblue3	24.8	3.85e08	*run-time exceeded*			> 7200	7.5	6.64e07	3.46e08	5342.6	0.0	3.74e08	4.09e08	1211.7
	bigblue4	14.9	9.27e08	*run-time exceeded*			> 7200	*crashed*				0.0	3.98e07	9.41e08	1316.1

CHAPTER 5

Improving Detailed Placement with Directed Moves

5.1 Overview

Detailed placement for standard cell and mixed-size designs has received substantial attention in the academic literature. During detailed placement, iterative techniques are employed to "undo" the damage to wire length (or other placement objectives) caused by "snapping" cells into non-overlapping locations during legalization. Moreover, detailed placement can be used to improve upon the final quality—as global placements are typically obtained without attention to local details, and since approximations are made in the cost functions for run-time reasons, some improvement in quality can be wrought by locally replacing cells.

Typically, detailed placement techniques pass a "sliding window" over the design and rearrange subsets of cells within the window to minimize local wire length. Optimal cell rearrangement has been noted to yield better results than heuristic strategies [23] because modern global placements already tend to be very good. The commonly-held belief is that heuristic methods are not as good at improving wire length because they can easily become trapped in local minima. Moreover, it is believed that stochastic search-based methods, like simulated annealing [75], can require too much run-time and are, therefore, impractical for large circuits. On the other hand, the traditional branch-and-bound techniques [23] can only optimize single-row windows containing less than 9 (or so) cells.

While it is clear that better local optimization can improve the quality of the entire placement,

there is a need to design detailed placement strategies which can yield good improvement in final wire length with only a minimal impact on overall run-time.

This chapter presents two approaches to solving the problem of detailed placement. The first technique is a dynamic programming formulation that can be used to optimize cells across multiple standard cell rows. Second, an annealing-based detailed placement strategy is described, and the effectiveness of directed moves within the context of such an annealer is explored.

The rest of this chapter is organized as follows. Section 5.2 describes the strategies for optimal wire length-based placement, and Section 5.3 examines the implementation of a simulated annealing-based detailed placer for standard cells.

5.2 Optimal Multi-Row Improvement Using A* Search

Since single-row branch-and-bound has been used successfully in `Capo` for years, it is expected that multi-row techniques could yield even further improvement. Unlike single-row branch-and-bound, however, multi-row methods cannot be implemented efficiently with a "right-edge" stack, as cells may be placed in either of two dimensions. To improve efficiency, this chapter presents a constrained version of a generic multi-row placement algorithm: rather than ripping up and determining where to replace cells, as in single-row branch-and-bound, the technique proposed here only allows cells to *exchange* positions. That is, cells are placed (optimally) only in those positions formally occupied by another cell (of the same size) in the sub-problem. This constraint preserves the legalization of the circuit, as well as the congestion metric, as no overlap is reintroduced.

Using this constraint, the multi-row strategy can be transformed into a dynamic programming instance, employing an A* search. The pseudocode for the generic A*-based placement method is presented in Figure 5.1. In each step, the algorithm tests the actual cost ("g") of assigning the next available cell in the sub-problem to an empty location. This creates a *child* of the current placement, as shown in line 15. A heuristic estimate h (given in line 17) is then computed which ranks each candidate by an estimate of the best wire length for the remaining nodes. The algorithm then visits each candidate configuration based on the best heuristic estimate. As cells are assigned to locations, they are fixed in place, and the algorithm proceeds to the next candidate.

The cost g is computed in a similar fashion as the costing strategy for assigning a cell μ to location λ, as in [44]. That is, the contributions to the HPWL of all nets attached to cell μ are summed, and the value is expressed as an incremental cost relative to the total cost of the candidate placement.

The quality of the heuristic estimation function directly impacts the speed of convergence of the A* approach. If the heuristic never overestimates the wire length, it is said to be an admissible

```
    Procedure: MULTI-ROW A* IMPROVEMENT
    Input: A set of cells to place
    Variables: An open and closed queue of candidate configurations
 1  begin
 2    initial_state ← empty; //No cells are placed initially.
 3    open_queue.insert( initial_state );
 4    while open_queue is not empty do
 5      //Grab the best candidate, and with it, the next empty location.
 6      curr_state ← open_queue.remove_front();
 7      if all cells in curr_state have been placed then
 8        return ; //Finished successfully.
 9      fi
10      closed_queue.insert( curr_state );
11      //Test each unplaced cell in this location.
12      child ← enumerate( curr_state );
13      for each i = 1...child.size() do
14        //Compute the actual increase in wire length for this assignment.
15        child[i].g ← compute_g( child[i] );
16        //Estimate the wire length required to place all remaining cells.
17        child[i].h ← compute_h( child[i] );
18        if child[i] exists in open_queue then
19          if child_in_open.g > child[i].g then
20            Replace child_in_open with child[i];
21          fi
22        else if child[i] exists in closed_queue then
23          if child_in_closed.g > child[i].g then
24            closed_queue.erase( child[i] );
25            open_queue.push( child[i] );
26          fi
27        else
28          open_queue.push( child[i] ); //First visit.
29        fi
30        i ← i + 1;
31      od
32      Sort open_queue using the sum of g and h;
33    od
34  end
```

Figure 5.1: Pseudocode for the multi-row, A*-based placement algorithm. Note that, because the heuristic underestimation function presented in this chapter decreases monotonically as more cells are placed, it is not necessary to check the "closed" queue. This check is merely provided in the pseudocode for completeness.

function which will always find an optimal solution.[1] The best possible heuristic will lead to the tightest possible bounding.

The algorithm to compute the heuristic estimate cost h is shown in Figure 5.2. To estimate the cost of the remaining cells in a candidate placement, the incremental costs of assigning each unplaced cell μ_i to each empty location λ_j are computed and summed. The key to this approach is that, when estimating remaining wire length, cells may be costed in *any* location, even if doing so would cause more than one cell to be costed in the same spot. As a result, the cost h is ensured to monotonically *underestimate* the actual wire length.

Empirical evidence suggests that h usually comes very close to approximating the actual wire length of the final (optimal) placement; therefore, the A* approach, on a whole, requires relatively few steps to place each subset of cells. While no proof of the monotonicity or underestimation of h is provided, computer simulations on more than 1 billion sub-problems have shown that h always underestimates the actual wire length of the placement, and that it decreases monotonically as more nodes are fixed in place.

5.2.1 Experimental Results

To test the quality of the A* strategy discussed in this chapter, `C++` code was integrated into the `Whim` tool (cf. Chapter 4). The `Capo` placement tool was used to produce legalized, HPWL-optimized placements, and the quality of improvement for these placements was observed. To broaden the comparisons, the single-row branch-and-bound technique from [24], a greedy (same-size) swapping method (akin to [27]), and a linear assignment method similar to [33] were also developed.

These four improvement techniques focus on slightly different levels of "locality". The greedy approach uses a large-scale binning strategy to gather up to 2000 cells per bin, where it then performs random pair-wise interchanges of same-sized cells. The linear assignment approach has been found to work best with subsets of approximately 20 cells, which is per [44], but unlike [33]. This number offers a good trade-off between performance and accuracy in modelling the assignment costs for each cell. On the other hand, empirical tests have found that the A* method can efficiently handle up to 8 cells. A localized window is used to gather cells for the problem, although it should be noted that, for the greedy, A*, and linear assignment approaches, this is not a strict requirement for proper operation.

The Peko benchmark suite [28] was used to compare the efficacy of all four detailed improvement methods. This benchmark offers several advantages to help in assessing the quality of the

1 If the estimate were to simply return zero, it would never overestimate the cost of a placement. In this case, the A* method would effectively implement Dijkstra's algorithm.

```
   Procedure: COMPUTE_H
   Input: A candidate placement
1  begin
2    cost_h ← 0;
3    for each unplaced node μ_i ∈ μ_0, ..., μ_n do
4      thisNodesBestCost ← ∞;
5      for each unfilled location λ_j ∈ λ_0, ..., λ_n do
6        if λ_j is not the same size as μ_i then
7          continue ;
8        fi
9        c_{μ_i λ_j} ← cost of assigning μ_i to λ_j; //Increment in HPWL.
10       if c_{μ_i λ_j} < thisNodesBestCost then
11         thisNodesBestCost ← c_{μ_i λ_j};
12       fi
13     od
14     cost_h ← cost_h + thisNodesBestCost;
15   od
16   return cost_h;
17 end
```

Figure 5.2: Pseudocode for computing a heuristic underestimation of the remaining wire length, given a candidate placement with some fixed and some unfixed cells (as well as a sufficient number of unfilled locations into which to place the unfixed cells).

four detailed placement approaches. First, each Peko circuit has a known optimal wire length; thus, it is possible to quantify the exact improvement of any particular algorithm. Second, all Peko circuits employ same-size cells, which permits the maximum number of cell-to-location matches for the A* and linear assignment approaches. Empirical evidence has shown that, with varying cell widths, these techniques do not find closely-placed cells of similar sizes; consequently, the quality of these methods can suffer on more realistic designs.

The statistics for the Peko circuits are presented in Table 5.1. The `Capo` placement tool, which was used to acquire legalized placements of these circuits, was run on a Linux-based Pentium 2.8 GHz computer. It should be noted that `Capo` made extensive use of "row ironing"—single-row, optimal branch-and-bound on subsets of up to 9 cells after placement. Thus, the post-legalized results presented here have already had some detailed improvement applied.

The four improvement methods implemented for this chapter were run on the same Pentium 2.8 GHz computer. The linear assignment method was configured to operate on subsets of up to 20 cells, the A* approach on subsets of up to 8 cells, and the single-row branch-and-bound on subsets of up to 6 cells. These sizes were determined empirically to offer the best trade-off

Table 5.1: Circuit statistics for Suite 3 of the Peko benchmarks. Wire length values are reported using HPWL (divided by 10^6), and CPU times are reported in seconds.

Circuit	Cells	Pads	Nets	Rows	Optimal WL	Capo	
						WL	CPU
peko01	12506	488	14111	113	0.822	1.436	39
peko02	19342	608	19584	140	1.27	2.353	62
peko03	22853	660	27401	152	1.51	2.737	78
peko04	27220	718	31970	166	1.76	3.301	97
peko05	28146	732	28446	169	1.95	3.660	104
peko06	32332	784	34826	181	2.07	3.798	114
peko07	45639	932	48117	215	2.89	5.623	183
peko08	51023	984	50513	227	3.15	5.775	202
peko09	53110	1004	60902	231	3.65	6.963	216
peko10	68685	1144	75196	263	4.75	9.725	306
peko11	70152	1154	81454	266	4.72	9.087	319
peko12	70439	1156	77240	266	5.02	9.590	336
peko13	83709	1260	99666	290	5.89	11.34	395
peko14	147088	1672	152772	385	9.03	17.44	789
peko15	161187	1748	186608	402	11.6	23.53	1029
peko16	182980	1864	190048	429	12.5	23.40	1163
peko17	184750	1872	189581	431	13.5	25.18	1220
peko18	210341	1998	201920	460	13.2	25.95	1320

of performance and quality. Each detailed placement algorithm was set to perform up to 35 passes. If the total improvement was less than 0.10% after every 5 passes of a given algorithm, the improvement strategy was stopped. In practice, no more than 5 passes of these algorithms were typically required.

Table 5.2 compares each of the four improvement strategies when run by themselves on the benchmark suite. For each circuit, the number of detailed improvement passes, CPU time (in seconds), and the ratio of HPWL after improvement to the HPWL prior to improvement. Results for the single-row branch-and-bound and greedy swapping methods are also presented.

The single-row branch-and-bound method yielded very little additional improvement over that of the legalized `Capo` placements. This was expected, since `Capo` applied one-row branch-and-bound to its placements. The A* and linear assignment techniques, on the other hand, offered an improvement of 2% on top of `Capo`. The performance for both methods was reasonable; this

indicates that the A* heuristic estimate offers tight bounds to the placement sub-problems.

It is worth noting that, despite the improvement reported here, the A* and linear assignment techniques are applicable only to same-sized cells, and thus the amount of improvement reported in Table 5.2 should be treated as a "best case" scenario. Modern standard cell designs typically feature a wide variety of cell widths; consequently, on circuits like those found in the ICCAD04 suite [6], the improvement wrought by the A* and linear assignment methods are negligible because they are unable to adequately explore the solution space. This suggests that a better, more robust technique is required.

5.3 Annealing-Based Detailed Placement

While the methods outlined in Section 5.2 do not increase placement density, they also do not help to improve it. These techniques are constrained by the fact that they only operate on same-sized cells, and do so within localized windows. Furthermore, the objective functions considered by these approaches are based solely on HPWL minimization—at least in the case of the A* technique, it might not be possible to enhance the method to account for additional objectives such as congestion or timing.

Given the advances in annealing-based placement for FPGAs, as described in Chapter 3, it seemed reasonable to employ annealing for detailed placement of standard cells. Annealing offers numerous advantages compared to other detailed placement strategies; chiefly, it can model complex objective functions, and its run-time behaviour can be well-controlled (through parameters such as `inner_num`, `inner_exp`, and starting temperature).

These advantages have always been thought to come at a cost, however: it has long been held that annealers scaled poorly with circuit size. The results presented later in this section contradict these beliefs, showing, instead, that annealing is quite competitive in terms of run-time. In addition, it is shown that the technique is effective in terms of improving placement quality, particularly for designs with near-uniform cell widths.

Much of what makes the annealer, in this work, so effective stems from the engineering and implementation details. While the general concept of such a placer is well-understood, it is the implementation of key aspects of the tool which ultimately determines its usefulness. It is also worth noting that, by using a simulated annealer, one can directly optimize multi-part cost functions (e.g., the ISPD06 cost function, which incorporates both an overlap metric and HPWL) without having to trade-off between competing objectives in multiple, separate placement phases as is often done in the literature (e.g., [128]).

The remainder of this chapter considers the details of an annealing based detailed placement

Table 5.2: Results for the linear assignment and A* improvement methods when using the grid binning strategy. All HPWL values have been divided by 10^6, and CPU times are reported in seconds. The average CPU time represents the average amount of time spent optimizing *per cell*, whereas the CPU value on each line represents the time spent optimizing the design .

Circuit	A* Method		Linear Assignment Method		Single-Row B&B		Greedy Swapping	
	CPU	WL Ratio	CPU	WL Ratio	CPU	WL Ratio	CPU	WL Ratio
peko01	30	0.973	37	0.978	5	0.999	57	0.985
peko02	49	0.977	44	0.977	6	0.999	57	0.988
peko03	55	0.976	51	0.981	8	0.999	96	0.985
peko04	48	0.977	80	0.982	9	0.999	108	0.987
peko05	73	0.973	84	0.969	22	0.996	91	0.985
peko06	78	0.977	70	0.980	11	0.999	128	0.986
peko07	130	0.978	99	0.981	18	0.999	122	0.989
peko08	130	0.977	150	0.979	21	0.999	154	0.988
peko09	141	0.976	117	0.979	19	0.999	160	0.988
peko10	140	0.977	201	0.978	29	0.999	224	0.989
peko11	138	0.976	153	0.979	25	0.999	203	0.988
peko12	189	0.976	208	0.975	31	0.999	400	0.986
peko13	172	0.976	246	0.979	37	0.999	279	0.988
peko14	279	0.978	422	0.979	59	0.999	420	0.989
peko15	323	0.979	356	0.980	76	0.999	603	0.989
peko16	381	0.976	543	0.975	78	0.999	1070	0.986
peko17	532	0.974	547	0.972	79	0.999	753	0.987
peko18	406	0.978	618	0.976	86	0.999	676	0.988
Averages	130 μs	0.976	147 μs	0.978	83 μs	0.999	163 μs	0.987

strategy.[2]

5.3.1 Implementation Details

The standard cell annealer in this work was based on `KPF` (cf. Chapter 3), which is, itself, loosely based on `VPR`. The standard cell annealer possesses all of the features of its FPGA counterpart: the ability to move multiple cells at once ("multi-moves"), a wire length-driven directed move based on median improvement, and the concept of move effectiveness to automatically choose the move most likely to yield improvement.

The standard cell annealer is not timing driven; instead, it optimizes the ISPD 2006 objective function, or, in the case of designs without the need for congestion minimization, it optimizes pin-to-pin HPWL. The scaled_overflow term in the objective function is computed using a discrete

2 Note that the results in subsequent sections focus entirely on simulated annealing-based placement, and not on the aforementioned strategies based on A* or linear assignment.

binning strategy which is very fast in practice.

For performance reasons, during the anneal, pin offsets are only considered for the largest $\approx 10\%$ of standard cells in a design (and even then, only if the standard cells exceed pre-defined size tolerances). The reason for this constraint is purely based on run-time: if pin offsets were to be considered for all standard cells, additional run-time would be required to verify when cells with pin offsets are moved, and this could minimize the effectiveness of the bounding box caching mechanism.

5.3.2 Overlap and Legality

In its earliest implementation, `Timberwolf` [101], permitted standard cells to overlap during placement, preferring instead to penalize overlaps as the temperature was decreased. As recognized in [110], such a formulation can be difficult to tune, as it can be challenging to trade-off overlap minimization, temperature, and cost function optimization. It is also unclear whether such a formulation would work in modern circuits which possess millions of placeable cells, feature large variations in cell widths, and have many fixed obstacles (such as pre-placed macrocells) which can make it difficult for cells to move outside of overlapping areas.

The approach employed in this thesis is to maintain legality throughout the anneal. After global legalization (cf. Chapter 4), standard cells are snapped to their nearest, legal site locations. All subsequent moves made during the anneal preserve legality. This requirement ensures that designs are always feasible, and that that the cost function can be computed precisely[3] since the (legalized) module locations are known. On the other hand, additional logic is required during the anneal to ensure that moves do not reintroduce overlap.

To this end, two types of strategies are described for preserving the legality of standard cell placements. The first strategy is based on a swap operation: when a source cell is requested to be placed in a target location, the contents of all cells at the target location (which could overlap with the source cell) are gathered and tested to see if they fit in the source location. Only those swaps which maintain legality are allowed to proceed (i.e., to have their costs computed).

The second strategy for maintaining legality is based on the ripple approach in Section 3.3.3. For FPGA placement, the ripple move was found to be more costly in terms of run-time, but very effective at minimizing the amount of displacement, and therefore, minimizing HPWL. The concept of a ripple move for standard cell annealing is similar (in that cells are rippled toward empty locations), though complicated by the presence of fixed obstacles and varying cell widths.

The pseudocode for the standard cell ripple-move is shown in Figure 5.3, and can be described

3 That is to say, unlike [110], there is no need to *estimate* the impact of a change in the placement on the cost function.

as follows. The technique operates on a data structure called a `QueueElement`, which is effectively a linked list that tracks the assignment of cells (from one bin to the next, and so forth). It is worth noting that the `QueueElement` stores only the set of feasible assignments and the assignment currently being processed. For example, a `QueueElement` qualitatively models the notion of "moving a node A from site (10,1) to site (5,3), then node B formerly at site (5,3) to the new site (5,4), and then node C formerly at (5,4) to an as-of-yet-untested site (8,4)". The ripple legalizer determines the chain of source-to-destination moves by searching radially outward from the initial destination point as long as the total displacement is less than the distance defined by the initial source node to its initial destination site.

A `QueueElement` is created for the initial source node and destination bin. This `QueueElement` is pushed to a priority queue which is sorted based on the total displacement of the chains specified by each `QueueElement`. While the queue is not empty, the maximum number of iterations is not exceeded, and the maximum displacement is not exceeded, the best queue element—the one with the least total displacement thus far—is selected for "exploration".

When a `QueueElement` is being "explored", it must be tested to ensure that the current assignment—in the above example, the assignment of node C to site (8,4)—does not overlap with other sites that may have been considered in the rippling chain, and that there is, indeed, enough room for C to fit at site (8,4). If these criteria are not satisfied, the queue element is "split" into four separate queue elements, each one following a different site to the left, up, down, and right of site (8,4). These new `QueueElements` are pushed to the priority queue as long as their new destination sites have not yet been visited.

On the other hand, if the criteria *are* satisfied, then site (8,4) is examined to see if it is empty. If it is empty, the list of assignments modelled by the `QueueElement` can be completed (because node C can be moved into the empty site), and the rippling is complete. If, instead, there are nodes at site (8,4), four new `QueueElements` must be created—one for each direction adjacent to site (8,4)—and pushed to the priority queue, where the rippling repeats with the nodes from (8,4).

5.3.3 Moves and Effectiveness

Based on the analysis conducted in Chapter 3, three types of moves were implemented for the standard cell annealer: a random move coupled with the simple legalizer, a median improvement move coupled with the simple legalizer, and a median improvement move coupled with the ripple legalizer.

Practically-speaking, the implementation of the median improvement move is identical to the discussion in Chapter 3. Given a randomly-chosen source, the median improvement heuristic is employed to find a target location. Then, either the simple or the ripple legalizer is invoked to

```
   Procedure: RIPPLE LEGALIZE
   Input: An initial source node, n_i and destination location, d_i
   Variables: A queue, q, which contains the current working QueueElement
   Return: True for success, false if a rippling strategy could not be found
1  begin
2    initialDisp ← distance of the displacement from n_i to d_i;
3    create a QueueElement for n_i and d_i, and push to q;
4    while q is not empty nor number of iterations exceeded do
5      qelem ← pop element in q with the least total displacement;
6      if the displacement of qelem is more than initialDisp then
7        return false;
8      fi
9
10     if the last assignment specified by qelem would not overlap with the rippling path or
11       the destination site is not feasible then
12
13       create 4 new QueueElements with the same assignment history but
14         whose destination is shifted one unit left, right, up, down;
15
16       push these 4 elements to q if the new destinations have not yet been visited;
17       continue
18     fi
19     gather the nodes at the destination site specified in qelem;
20     if there are no nodes at the destination site then
21       // Rippling succeeded
22       return true;
23     else
24       create 4 new QueueElements, appending the nodes from the destination site, and setting
25         the next site to explore as the bin to the left, right, up, and down of the current destination;
26
27       push these 4 elements to q if the new destinations have not yet been visited;
28     fi
29   od
30 end
```

Figure 5.3: Pseudocode for the standard cell rippling strategy.

ensure that the move maintains legality. If the move is legal, its cost is computed and it is either accepted or rejected.

Random moves are also employed in a manner similar to the FPGA annealer. That is, given a randomly-chosen source cell, a range-limited window is used to narrow the search for the destination location. The dimensions of this window are shrunk as the temperature decreases. One slight difference with the FPGA tool, however, is that the window represents a *multiple* of the number of cell heights and widths for the given (source) cell. Consider the following illustrative example. If the random window size specifies 12 units by 12 units, and a source cell is provided which is 5 standard cell units in width, the random move will search within a window measuring 60 units in width by 12 units in height, centred about the source cell. This ensures that only a single range-limited window is required for designs which might possess widely-varying cell widths.

The concept of move effectiveness is also employed in the standard cell annealer, and is identical to the implementation described in Chapter 3. Because the move effectiveness metric can account for run-time, it is particularly useful at balancing the frequency with which moves based on the ripple legalizer and simple legalizer are employed; this is vital to good run-time performance because the ripple legalizer is computationally more intensive than the simple technique. Move effectiveness is also useful for terminating the anneal, especially when the anneal is performed in a low- (or zero-) temperature regime.

5.3.4 Experimental Results

The simulated annealer described previously was implemented in `Whim` (cf. Chapter 4). Several tests were conducted using the ISPD 2006 [91] and ICCAD04 [6] benchmarks to establish the effectiveness of this annealer and its directed moves. The ISPD 2006 suite can be viewed as an extension of the ISPD 2005 suite, featuring more designs and incorporating the concept of a "target density" for congestion minimization. The characteristics of these circuits are shown in Table 5.3. Contrarily, designs in the ICCAD04 suite do not possess target densities, but feature a set of widely-varying, movable macrocells; characteristics for this suite are summarized in Table 5.4. In the case of the ISPD 2006 suite, the scripts provided by the ISPD contest organizers were employed to compute the overlap and HPWL for all tools considered in this work, whereas the `Whim` tool was used to measure the pin-to-pin HPWL for all designs considered in the ICCAD04 suite. In all cases, `Capo` was employed to ensure that the results were legal (and, in all cases, this was found to be true).

In the following discussion, several tests were conducted in which global placements were first produced by `mPL` [27], and the legalizers and detailed placement strategies of `mPL`, `NTUplace3`, and `Whim` were compared. `mPL` was chosen to produce the global placements because it placed first

Table 5.3: Characteristics of the ISPD 2006 suite. Note that these designs do not possesses movable macrocells.

	General Statistics		Standard Cell Widths			Target
Circuit	**Cells**	**Nets**	**Min**	**Max**	**Median**	**Density**
adaptec1	210904	221142	3	77	13	0.6
adaptec2	254457	266009	3	75	8	0.6
adaptec3	450927	466758	3	80	10	0.6
adaptec4	494716	515951	3	80	10	0.6
adaptec5	842482	867798	3	80	8	0.5
bigblue1	277604	284479	3	43	11	0.6
bigblue2	534782	577235	3	104	9	0.6
bigblue3	1093034	1123170	3	104	8	0.6
bigblue4	2169183	2229886	3	104	8	0.6
newblue1	330073	338901	3	75	8	0.8
newblue2	436516	465219	6	212	20	0.9
newblue3	482833	552199	3	104	15	0.8
newblue4	642717	637051	3	106	10	0.5
newblue5	1228177	1284251	3	104	8	0.5
newblue6	1248150	1288443	3	104	10	0.8
newblue7	2481372	2636820	3	104	8	0.8

in the ISPD 2006 contest [91] in terms of the combined cost function (wire length and overlap), and (narrowly) placed second in the category of wire length alone. In some tests, both global and detailed placements were produced by `mPL`, and `Whim`'s annealing strategy was employed to improve upon them. All designs were executed on a Pentium 2.8 GHz Linux machine with 4 GB of RAM. Run-times were measured in CPU seconds. All results reported from `Whim`'s simulated annealer were averaged over three different random seeds.

5.3.4.1 Tests on Global Placements

Using the global placements generated by `mPL`, the legalizer and simulated annealer built into `Whim` were compared to other legalization and detailed placement tools in an effort to answer the following questions:

1. How effective are directed moves on standard cell designs?

Table 5.4: Characteristics of the ICCAD04 suite.

	General Statistics					Standard Cell Widths		
Circuit	**Cells**	**Macros**	**Nets**	$\textbf{Area}_{\textbf{Largest}}(\%)$	$\frac{\textbf{Area}_{\textbf{Largest}}}{\textbf{Area}_{\textbf{Smallest}}}$	**Min**	**Max**	**Median**
ibm01	12260	246	14111	6.4	8412.0	2	46	8
ibm02	19071	271	19584	11.4	30053.0	2	100	6
ibm03	22563	290	27401	10.8	33088.0	2	146	6
ibm04	26925	295	31970	9.2	26600.0	2	162	8
ibm05	28146	0	28446	0.0	10.0	2	20	8
ibm06	32154	178	34826	13.6	36358.0	2	56	4
ibm07	45348	291	48117	4.8	17570.0	2	146	6
ibm08	50722	301	50513	12.1	50874.0	2	100	6
ibm09	52857	253	60902	5.4	29716.0	2	166	6
ibm10	67899	786	75196	4.8	71316.0	2	160	10
ibm11	69779	373	81454	4.5	29711.0	2	166	6
ibm12	69788	651	77240	6.4	74266.5	2	46	12
ibm13	83285	424	99666	4.2	33098.5	2	146	6
ibm14	146474	614	152772	2.0	17867.5	2	152	6
ibm15	160794	393	186608	11.0	125580.0	2	184	8
ibm16	182522	458	190048	1.9	31104.0	2	52	8
ibm17	183992	760	189581	0.9	12446.5	2	160	12
ibm18	210056	285	201920	1.0	10150.0	2	46	8

2. Given that `Whim` attempts to retain global placement quality (by minimally perturbing placements during legalization), how effective is it when coupled with annealing-based detailed placement for the ISPD 2006 designs?

3. Can a low-temperature anneal improve upon existing, final solutions from `mPL` or `NTUplace3`, and if so, what are the run-time implications?

To measure the effectiveness of directed moves during annealing-based placement, a similar methodology as that described in Chapter 3 was employed. Each global placement in the ISPD 2006 suite was legalized and detail-placed with and without directed moves for 3 separate random seeds, using four configurations which varied the run-time and effort-level of the annealer. These four configurations are summarized in Table 5.5, and were derived after a series of empirical studies which suggested that these configurations led to good quality and run-time trade-offs.

Table 5.5: Parameter configurations used for testing the annealer. Configuration A is the slower, higher-quality annealer setting, while Configuration D is the fastest setting. These configurations were chosen based on empirical testing.

Configuration	Description
A	`inner_num` = 1 `inner_exp` = 1.15 `random window size` = 12×12 `initial temp` set to achieve $\approx 20\%$ acceptance rate
B	`inner_num` = 1 `inner_exp` = 1.10 `random window size` = 10×10 `initial temp` set to achieve $\approx 20\%$ acceptance rate
C	`inner_num` = 1 `inner_exp` = 1.05 `random window size` = 8×8 `initial temp` set to achieve $\approx 15\%$ acceptance rate
D	`inner_num` = 1 `inner_exp` = 1.0 `random window size` = 6×6 `initial temp` set to achieve $\approx 10\%$ acceptance rate

The run-times and resultant quality (as measured using the ISPD 2006 cost function) were gathered for each design in the suite. The costs and run-times were then normalized relative to the fastest, lowest-quality configuration. The resultant graph illustrating the effectiveness of annealing with directed moves (versus an anneal without directed moves) is shown in Figure 5.4. It is worth emphasizing that each point in this graph represents the normalized geometric mean of the quality and run-time of the *entire* ISPD 2006 suite for a particular configuration. From this graph, the trend is clear—as in Chapter 3, the directed moves consistently dominate in quality (for the same amount of run-time) compared to an anneal performed without directed moves.

Another experiment was conducted to establish how well `Whim`'s legalization and detailed improvement strategy compared to the state-of-the-art methods employed by `mPL` and `NTUplace3`. For this test, the global legalization strategy described in Chapter 4, coupled with the annealer using Configuration A (cf. Table 5.5) were used to legalize and improve `mPL`-generated global placements. The results from `Whim` were then compared to both `mPL` and `NTUplace3`, and are

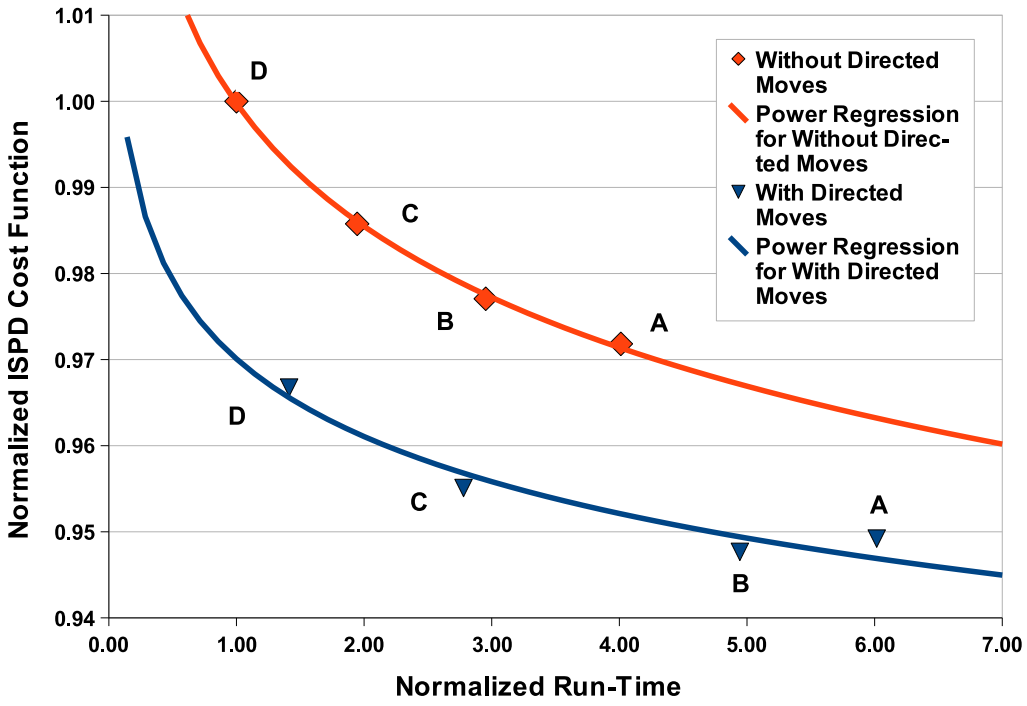


Figure 5.4: Illustration of the dominance of directed moves on the ISPD 2006 suite.

summarized in Table 5.6. `Whim` achieved a 4% improvement, on average, versus these existing tools, consistently producing placements with not just better HPWL but also less overlap.

5.3.4.2 Tests on Detailed Placements

Another experiment was conducted using the ISPD 2006 test suite in which the annealer (with directed moves enabled) was employed to improve upon already-optimized, legalized placements produced by `mPL`. This test sought to answer the question of whether or not an anneal (with $T > 0$) would be capable of escaping a locally-minimal solution and achieve a meaningful improvement in quality. The results achieved by the annealer using Configuration A—the most run-time intensive setting—are shown in Table 5.7. The annealer was able to improve upon existing `mPL` placements by 4% on average. While the run-time for the technique was higher than `mPL`'s legalization strategy, it is worth noting that it was still less than that required for the global placement. Moreover, for this test, the annealer was executed using the setting with the largest run-time—thus, given the trend shown in Figure 5.4, the run-time of the annealer could be easily halved (by adjusting the appropriate parameters) with only a $\approx 1\%$ loss in quality.

It is also worth noting that the run-times for the annealer are generally consistent and scalable

Table 5.6: Quality of Whim's legalization and detailed placement versus mPL and NTUplace3. "WL" represents the HPWL divided by 10^6, "SO" represents the ISPD 2006 "scaled overlap" metric, and "TC" represents the total cost function.

Circuit	mPL			NTUplace3			Whim + Anneal			Whim TC vs		Whim CPU vs	
	WL	SO	TC	WL	SO	TC	WL	SO	TC	mPL	NTU	mPL	NTU
adaptec1	90.25	1.39	91.51	90.37	2.07	92.24	88.88	0.34	89.18	0.97	0.97	1.7	5.24
adaptec2	102.15	1.43	103.62	102.48	1.79	104.31	99.63	0.36	99.99	0.96	0.96	1.96	4.61
adaptec3	238.11	0.84	240.12	236.6	1.34	239.77	231.11	0.32	231.84	0.97	0.97	2.05	4.61
adaptec4	209.13	0.6	210.4	208.3	0.95	210.27	202.58	0.23	203.05	0.97	0.97	1.98	4.73
adaptec5	424.27	0.92	428.17	421.07	1.92	429.16	409.7	0.46	411.59	0.96	0.96	2.24	3.5
bigblue1	114.56	1.18	115.91	113.85	1.66	115.73	112.74	0.29	113.06	0.98	0.98	1.77	4.59
bigblue2	164.38	1.22	166.39	163.27	1.69	166.04	159.12	0.41	159.76	0.96	0.96	1.76	3.59
bigblue3	414.91	0.63	417.51	410.66	1.36	416.24	397.75	0.3	398.95	0.96	0.96	2.66	3.99
bigblue4	912.09	1.27	923.7	905.07	1.45	918.16	880.19	0.52	884.77	0.96	0.96	2.36	4.75
newblue1	66.53	0.61	66.94	66.16	8.24	71.61	64.88	0.09	64.94	0.97	0.91	2.47	4.19
newblue2	199.05	1.39	201.82	199.12	0.6	200.31	197.98	0.11	198.2	0.98	0.99	2.77	7.59
newblue3	283.54	0.61	285.27	280.54	0.2	281.11	271.34	0.05	271.47	0.95	0.97	1.7	4.53
newblue4	293.47	1.55	298.02	290.23	2.8	298.37	281.55	0.65	283.37	0.95	0.95	2.01	3.65
newblue5	528.44	1.37	535.66	524.69	2.35	537.04	510.65	0.73	514.38	0.96	0.96	2.17	3.6
newblue6	516.37	1.31	523.12	515.64	0.56	518.51	504.27	0.2	505.3	0.97	0.97	2.55	6.1
newblue7	1073.1	1.11	1085.05	1070.44	0.5	1075.79	1043.91	0.23	1046.36	0.96	0.97	2.92	6.83
									Geomeans	0.96	0.96	2.16	4.64

relative to mPL. This stems from the fact that the annealer scales based on the VPR-inspired formula in Equation (2.15), and that in particular, for Configuration A, inner_exp was set to 1.15. By adjusting this parameter, one can trade-off better quality for better run-time. This flexibility allows the user to control whether the annealer scales linearly or at a weak power of the number of placeable objects in the design.

5.3.4.3 Commentary on Cell Diversity and Annealing

Qualitatively, it has been observed that maintaining legality in circuits with varying standard cell widths can be *very* hard—the annealer may "abort" many moves, as legality may not be achievable. It is worth considering whether or not, as cell widths become less diverse, the amount of improvement offered by a simulated annealing-based placer gets better. In other words, is it possible that simulated annealing is more effective for architectures with more-uniform cell widths (such as FPGAs and structured ASICs)?

The term "aborted", in this discussion, applies to a move which could not be costed for any reason. There are several explanations as to why a move may not be costed. The most common explanations are related to the inability of the simple or ripple legalizers to maintain the feasibility of the placement; some of these situations are illustrated in Figure 5.5. One of the most common

Table 5.7: Quality of `Whim`'s detailed placement at improving already-optimized placements produced by `mPL`. "WL" represents the HPWL divided by 10^6, "SO" represents the ISPD 2006 "scaled overlap" metric, "TC" represents the total cost function, and "GP" standards for "global placement".

Circuit	mPL			Whim's Annealer			Ratios		
	WL	SO	TC	WL	SO	TC	$\frac{\text{Anneal TC}}{\text{mPL TC}}$	$\frac{\text{Anneal CPU}}{\text{mPL CPU}}$	$\frac{\text{Anneal CPU}}{\text{mPL GP CPU}}$
adaptec1	90.25	1.39	91.51	88.70	0.50	89.14	0.97	1.49	0.45
adaptec2	102.15	1.43	103.62	99.13	0.49	99.61	0.96	1.67	0.41
adaptec3	238.11	0.84	240.12	230.15	0.40	231.07	0.96	1.75	0.26
adaptec4	209.13	0.60	210.40	202.19	0.27	202.74	0.96	1.77	0.25
adaptec5	424.27	0.92	428.17	408.12	0.67	410.87	0.96	1.93	0.54
bigblue1	114.56	1.18	115.91	112.46	0.43	112.94	0.97	1.53	0.60
bigblue2	164.38	1.22	166.39	158.70	0.56	159.59	0.96	1.58	0.49
bigblue3	414.91	0.63	417.51	397.54	0.40	399.12	0.96	2.36	0.72
bigblue4	912.09	1.27	923.70	875.51	1.15	885.57	0.96	2.13	0.78
newblue1	66.53	0.61	66.94	64.72	0.17	64.83	0.97	2.21	0.74
newblue2	199.05	1.39	201.82	194.44	0.32	195.07	0.97	2.08	0.48
newblue3	283.54	0.61	285.27	270.86	0.09	271.11	0.95	1.40	0.27
newblue4	293.47	1.55	298.02	280.52	0.93	283.14	0.95	1.76	0.51
newblue5	528.44	1.37	535.66	507.82	1.05	513.17	0.96	1.96	0.58
newblue6	516.37	1.31	523.12	501.81	0.59	504.75	0.96	2.31	0.78
newblue7	1073.10	1.11	1085.05	1034.44	0.72	1041.93	0.96	2.45	0.70
						Geomeans	0.96	1.87	0.5

reasons for an aborted move occurs when the simple legalizer tries to move a cell into a position which overlaps with a larger cell (as in the case of node 1, in the figure) or a set of cells (in the case of node 0), but there is insufficient room for the reverse "swap". Alternatively, cells could be moved inside unusable areas (such as inside macrocells or off the edge of the chip). Another common source of aborted moves occurs while rippling: if an empty space could not be found quickly enough, the annealer will abort the attempt. (This is done, by design, to maintain reasonable run-times.) Additionally, if the outward search performed during rippling would have incurred too much displacement, the ripple will be aborted. Although less common, it is possible that, due to the order in which nodes are rippled, too large a set of cells may be in the current `QueueElement` to find a feasible solution, and the search space will have been exhausted. It is also possible that a node is placed at its optimal wire length-minimizing location, so the target location computed by the median improvement move would overlap on top of the current cell position—this can lead to a move being aborted because no new (or different) candidate location may have been

found. Later in placement, as its window size shrinks, the random move can also "trap" small cells around larger cells, since there would be less room with which to move the small cells out from around the larger ones.

To address this question, four separate suites were employed: the first was the original ICCAD04 suite; next, the same ICCAD04 suite, but with standard cell widths divided by 5, as well as one with standard cell widths divided by 5 but the maximum size limited to 3 units; and finally, the ICCAD04 suite with all standard cell widths set to the same, unit size. In each case, the row layout was modified to maintain the same 10% whitespace and aspect ratio *across all designs*. `mPL` was used to produce wire length-optimized (legalized and improved) placements for all designs, and Configuration A (from Table 5.5) was employed to *further* improve the results. The cell statistics for the suites where site widths were shrunk by a factor of 5 are provided in Table 5.8.

Table 5.9 summarizes the results achieved across all four suites. From this chart, it is clear that, as the cells in the designs approached more uniform widths, the number of "aborted" moves decreased and the efficacy of the annealer improved substantially. Not only does the quality of the annealing-based improvement get better based on the characteristics of the circuit to which it was applied, it far exceeds the ability of typical, end-case optimal algorithms which, perhaps, are only effective on those circuits with widely varying widths. This observation offers a possible explanation as to why annealing remains the algorithm of choice for architectures with more uniform, smaller cell widths, like FPGAs and structured ASICs.

An analysis of the aborted moves reveals several interesting trends. The ratio of the number of attempted-to-aborted moves remains reasonably consistent throughout the anneal—while it may be slightly higher at the beginning, the ratio decreases by a few percent as the anneal progresses. This corresponds with the fact that some (but not the majority of) cells are in their optimal HPWL-minimizing locations near the end of placement. Also, as cells are placed closer together to minimize HPWL, the rippling moves and random window moves are less likely to find legal, non-overlapping positions.

The most common cause of aborted moves was found to be the reverse swap operation in the simple move legalizer—that is, the swap of the nodes from the target location back to the source's original position. Such moves are more difficult to legalize because the size of the nodes at the target location often exceed the space available at the source in designs with varying cell widths. This explains why there was a significant increase in the ratio of attempted-to-aborted moves as the cells were shrunk to unit width (since such aborts are not possible for unit-width circuits). Another common cause for aborted moves in these designs was due to the fact that the ripple legalizer could not find a valid rippling path which minimized the amount of displacement. This is particularly the case in these ICCAD04-derivative benchmarks due to the minimal amount of white space and

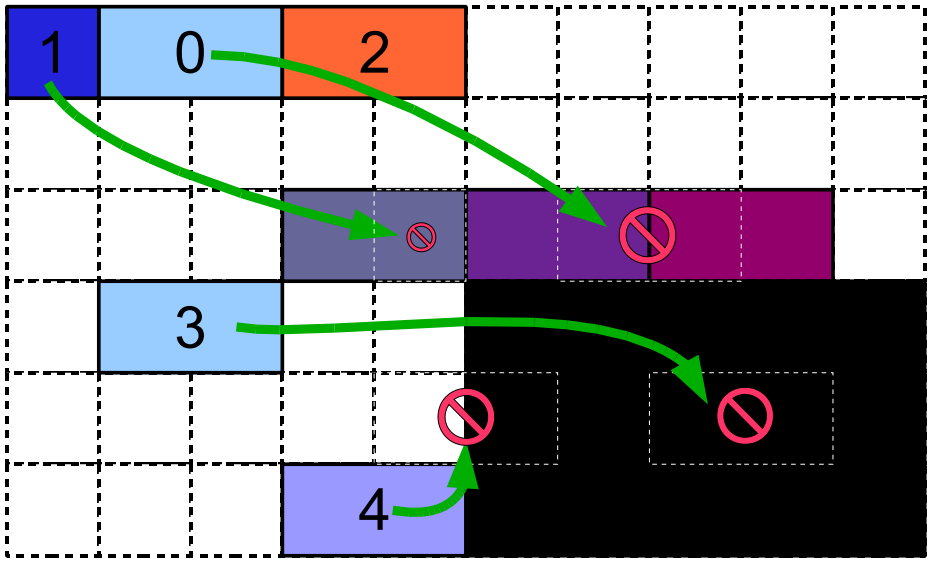


Figure 5.5: Illustration of aborted moves in the standard cell annealer.

large percentage of macrocell area.

It is important to consider that aborted moves are not necessarily detrimental to run-time—some sources for aborted moves have greater impact on run-time than others. For instance, the aborted moves most commonly seen by the annealer—the reverse swap in the simple legalizer—can be identified very quickly, which allows the annealer to attempt another move without much impact on run-time. On the other hand, the ripple move may spend ten times the amount of run-time as the simple legalizer only to discover, at the end of its processing, that it was unable to find a legal solution. In this case, the move effectiveness heuristic compensates for this processing time by lowering the probability with which such moves are chosen in the anneal.

5.4 Conclusion

This chapter described two new methods for improving placements—one based on a novel application of dynamic programming, and the other on simulated annealing. While the A*-based strategy proved effective at HPWL minimization, it was ineffective at minimizing more complex objectives (such as those incorporating congestion), and could not achieve meaningful improvement in designs which possess a plethora of cell widths.

To address this issue, a simulated annealer was incorporated into `Whim` and tested on a variety of modern designs. The annealing-based strategy proved to be very effective at optimizing the circuits in ISPD 2006 suite, especially when coupled with a directed move based on median placement and a ripple-move legalization strategy.

Table 5.8: Characteristics of the ICCAD04 suite with cell widths shrunk by a factor of 5, as well as the suite with cell widths shrunk by a factor of 5 but limited to maximum size of 3. The number of cells, nets, and whitespace remain the same as the original suite.

	Cell Widths for Suite					
	No Max Width			Max Width 3		
Circuit	Min	Max	Median	Min	Max	Median
ibm01	1	10	2	1	3	2
ibm02	1	20	2	1	3	2
ibm03	1	30	2	1	3	2
ibm04	1	33	2	1	3	2
ibm05	1	4	2	1	3	2
ibm06	1	12	1	1	3	1
ibm07	1	30	2	1	3	2
ibm08	1	20	2	1	3	2
ibm09	1	34	2	1	3	2
ibm10	1	32	2	1	3	2
ibm11	1	34	2	1	3	2
ibm12	1	10	3	1	3	3
ibm13	1	30	2	1	3	2
ibm14	1	31	2	1	3	2
ibm15	1	37	2	1	3	2
ibm16	1	11	2	1	3	2
ibm17	1	32	3	1	3	3
ibm18	1	10	2	1	3	2

When compared to existing tools, the annealer consistently achieved improvements over existing state-of-the-art methodologies, and was also capable of improving substantially upon already-optimized placements. This work also showed that simulated annealing is more effective at improving designs with similar cell widths due to the fact that more moves remain legal (i.e., fewer moves are aborted). This observation lends credence to the notion that annealing may be more effective at placing designs with more regular cells, such as those found in FPGAs or structured ASICs. Finally, an anneal with directed moves was shown to consistently dominate versus an anneal performed without directed moves, for the same amount of run-time effort.

Table 5.9: Results comparing the effectiveness of annealing on standard cell designs, as the width of standard cells is decreased. HPWL results are divided by 10^6. "Initial" HPWL refers to the optimized, legalized HPWL achieved by `mPL`, whereas "Final" HPWL is the HPWL after annealing-based improvement in `Whim`.

Design	ICCAD04				Shrunk by 5				Shrunk by 5, max 3				Shrunk to unit-width			
	HPWL			Moves	HPWL			Moves	HPWL			Moves	HPWL			Moves
	Initial	Final	$\frac{\text{Final}}{\text{Initial}}$	$\frac{\text{\# Attempts}}{\text{\# Aborts}}$	Initial	Final	$\frac{\text{Final}}{\text{Initial}}$	$\frac{\text{\# Attempts}}{\text{\# Aborts}}$	Initial	Final	$\frac{\text{Final}}{\text{Initial}}$	$\frac{\text{\# Attempts}}{\text{\# Aborts}}$	Initial	Final	$\frac{\text{Final}}{\text{Initial}}$	$\frac{\text{\# Attempts}}{\text{\# Aborts}}$
ibm01	2.22	2.22	1.00	0.69	1.49	1.46	0.98	1.12	1.63	1.53	0.94	1.29	1.94	1.62	0.84	19.48
ibm02	4.81	4.80	1.00	0.88	4.40	3.91	0.89	1.17	4.28	3.75	0.88	1.20	3.66	3.03	0.83	30.25
ibm03	6.72	6.55	0.98	0.64	4.42	4.28	0.97	0.90	4.90	4.24	0.87	1.15	4.01	3.73	0.93	66.13
ibm04	7.32	7.30	1.00	0.64	5.55	5.19	0.94	1.01	4.74	4.50	0.95	1.36	5.34	4.43	0.83	35.67
ibm05	9.39	9.38	1.00	1.07	4.68	4.64	0.99	1.52	4.66	4.62	0.99	1.58	3.15	3.05	0.97	1067.18
ibm06	5.73	5.72	1.00	0.61	4.45	3.99	0.89	1.43	3.84	3.52	0.92	1.74	4.56	3.81	0.83	54.06
ibm07	9.82	9.78	1.00	0.64	6.49	6.15	0.95	0.93	6.03	5.77	0.96	1.12	5.16	4.74	0.92	89.64
ibm08	11.94	11.94	1.00	0.91	8.52	8.10	0.95	1.18	10.06	8.91	0.89	1.26	7.26	6.74	0.93	133.11
ibm09	12.34	12.23	0.99	0.68	7.93	7.49	0.94	0.97	7.32	6.95	0.95	1.20	7.75	6.82	0.88	97.83
ibm10	27.75	27.59	0.99	0.77	22.20	21.46	0.97	1.05	23.23	21.36	0.92	1.48	27.51	23.06	0.84	21.31
ibm11	18.18	17.67	0.97	0.65	10.99	10.53	0.96	0.94	9.93	9.55	0.96	1.23	8.72	7.99	0.92	134.46
ibm12	31.92	31.82	1.00	0.74	21.80	21.43	0.98	1.11	21.81	21.33	0.98	1.72	23.62	21.42	0.91	35.32
ibm13	22.46	22.29	0.99	0.59	14.99	14.09	0.94	0.87	13.28	12.62	0.95	1.22	12.46	11.13	0.89	120.41
ibm14	35.70	35.41	0.99	0.74	20.96	20.13	0.96	1.09	21.01	20.01	0.95	1.32	16.68	15.57	0.93	121.54
ibm15	47.07	46.05	0.98	0.80	27.19	26.29	0.97	1.13	26.09	24.16	0.93	1.32	20.41	18.85	0.92	220.07
ibm16	55.60	55.46	1.00	0.79	35.76	34.64	0.97	1.09	40.23	37.64	0.94	1.24	31.37	28.46	0.91	93.66
ibm17	64.65	64.50	1.00	0.82	35.24	34.37	0.98	1.15	34.42	33.36	0.97	1.60	25.37	23.99	0.95	102.15
ibm18	42.10	42.10	1.00	0.93	22.26	21.74	0.98	1.46	22.17	21.65	0.98	1.57	15.98	15.20	0.95	542.03
Geomeans			0.99	0.75			0.95	1.10			0.94	1.35			0.90	90.26

CHAPTER 6

Conclusion

6.1 Summary

This thesis discussed a number of strategies for augmenting stochastic search techniques with non-random, "directed" moves. These moves were implemented within the context of various VLSI CAD algorithms, including FPGA placement, mixed-cell floorplan repair, and standard cell annealing.

In terms of FPGA placement, several types of directed moves were considered, and results were presented that showed significant improvements over simple, random moves for the same amount of annealing effort. It was shown that directed moves are useful in devices with lower utilization, and can reduce the statistical variability in placements, which can lead to more repeatable results. Furthermore, it was established that the benefits of directed moves cannot be achieved by changing the annealer's cost function. A zero-temperature approach for performing BLE operations was described, and a technique for measuring move effectiveness was also proposed.

Directed moves were also introduced within the context of a top-down strategy employing a combination of constraint graphs, linear programming, and a novel geometric shifting technique. This repair strategy moves only those features which are responsible for violating overlap constraints, thereby making it a versatile way to post-process the outputs of global floorplanners and placers. The effectiveness of this algorithm was quantified across a broad range of floorplans produced by multiple tools, producing placements with less movement, on average, than leading methods.

Finally, the concepts of both optimal and heuristic detailed placement were explored, with the notion of directed moves being examined in a simulated annealing-based standard cell placer.

While the optimal placement strategy produced satisfactory improvements in certain circumstances, it was ineffective at minimizing more complex objectives (such as those incorporating congestion), and could not achieve meaningful improvement in designs with varying cell widths. To address this issue, an annealer, augmented with directed moves, was introduced. The annealer achieved better results than existing state-of-the-art methodologies, and was even capable of improving upon already-optimized placements. Furthermore, this work showed that simulated annealing is more effective at improving designs with similar cell widths due to the fact that more moves remain legal (i.e., fewer moves are aborted). This observation lends credence to the notion that annealing may be more effective at placing designs with more regular cell widths, such as those found in FPGAs or structured ASICs.

6.2 Future Directions

While this thesis described practical methods for improving stochastic search techniques in VLSI CAD, it also laid the foundation for further research.

The directed moves considered in this work only begin to scratch the surface of what may be possible in annealing-based placement for FPGAs. While the `Domino` moves proved to be ineffective on a run-time basis, an open question remains as to whether or not a faster, more efficient implementation can be made which could optimize hundreds of cells at once, and if the concept of timing could be incorporated into the formulation. In [4], a quadratic formulation was employed as a type of directed move for minimizing wire length in the `Parquet` floorplanner; while not as accurate as the median placement strategy described in this work, it may offer better run-time scaling and may prove to be useful for both wire length and timing minimization. Power is fast-becoming an important optimization metric for modern FPGA CAD as well, and it may be possible to derive power-minimizing moves for annealing-based placement. Furthermore, it may be worthwhile to incorporate the concept of DRC-correctness into the BLE-based directed moves to minimize the number of moves which are rejected due to cluster infeasibility.

In terms of floorplan repair, there exists some room for future improvement. The modelling of standard cells and whitespace as soft (resizable) macros within the minimum movement floorplanner may allow it to legalize large cells without reintroducing as much overlap (i.e., by preventing it from accidentally moving a large macro on top of many standard cells). Furthermore, it is felt that there is room for improving upon the simplistic partitioning strategy that was described, perhaps by incorporating look-ahead partitioning and a more intelligent cutline placement mechanism.

With the advances put forth in this work, simulated annealing-based detailed placement may find a renewed interest in the academic community. To this end, wire length- and congestion-minimizing directed moves could prove invaluable for improving design quality. Faster approaches

for cell rippling (that is, maintaining cell legality while minimizing the displacement of the cells) may also be useful; for example, a legalization strategy which ripples via flow-based methods may be a worthwhile pursuit. An open question remaining in this thesis is that of how well an annealer which maintains legality during an anneal compares to an annealer which permits overlaps (but purges said overlaps via a penalty term in the objective function or through shifting). Perhaps a hybrid approach—where overlaps are occasionally reintroduced and then re-legalized—could lead to better quality or run-time scalability.

In summary, stochastic search techniques may well see a renaissance in the field of VLSI CAD; the future of module placement will hinge on new discoveries that build upon the knowledge of the past.

BIBLIOGRAPHY

[1] ACTEL CORPORATION. *ProASIC3 Flash Family FPGAs datasheet v1.1*. Actel, 2009.

[2] ADYA, S. N., CHATURVEDI, S., ROY, J. A., PAPA, D. A., AND MARKOV, I. L. Unification of partitioning, placement and floorplanning. In *Proceedings of ICCAD* (2004), pp. 550–557.

[3] ADYA, S. N., AND MARKOV, I. L. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proceedings of ISPD* (2002), pp. 12–17.

[4] ADYA, S. N., AND MARKOV, I. L. Fixed-outline floorplanning: Enabling hierarchical design. *IEEE Transactions on VLSI Syst. 11*, 6 (2003), 1120–1135.

[5] ADYA, S. N., AND MARKOV, I. L. Improving min-cut placement for VLSI using analytical techniques. In *Proceedings of IBM ACAS Conference* (2003), IBM ARL, pp. 55–62.

[6] ADYA, S. N., AND MARKOV, I. L. ICCAD04 mixed-size placement benchmarks. `http://vlsicad.eecs.umich.edu/BK/ICCAD04bench`, 2004. Current May 2009.

[7] AGNIHOTRI, A., YILDIZ, M. C., KHATKHATE, A., MATHUR, A., ONO, S., AND MADDEN, P. H. Fractional cut: Improved recursive bisection placement. In *Proceedings of ICCAD* (2003), pp. 307–310.

[8] AGNIHOTRI, A. R., ONO, S., AND MADDEN, P. H. Recursive bisection placement: Feng shui 5.0 implementation details. In *Proceedings of ISPD* (2005), pp. 230–232.

[9] ALPERT, C. J., CHAN, T., HUANG, D. J., KAHNG, A. B., MARKOV, I. L., MULET, P., AND YAN, K. Faster minimization of linear wirelength for global placement. In *Proceedings of ISPD* (1997), pp. 4–11.

[10] ALPERT, C. J., AND KAHNG, A. B. Recent directions in netlist partitioning: a survey. *Integr. VLSI J. 19*, 1-2 (1995), 1–81.

[11] ALPERT, C. J., NAM, G.-J., AND VILLARRUBIA, P. G. Free space management for cut-based placement. In *Proceedings of ICCAD* (2002), ACM Press, pp. 746–751.

[12] ALTERA CORPORATION. Stratix II device handbook v4.3. Data Sheet, Altera Corporation, January 2008.

[13] ALTERA CORPORATION. Quartus II handbook v9.0. Tech. rep., Altera Corporation, March 2009.

[14] BALDICK, R., KAHNG, A. B., KENNINGS, A., AND MARKOV, I. L. Function smoothing with applications to VLSI layout. In *Proceedings of ASPDAC* (1999), pp. 225–228.

[15] BELL, B. A., AND FEINER, S. K. Dynamic space management for user interfaces. In *Proceedings of User interface software and technology* (2000), pp. 239–248.

[16] BERAUDO, G. A path based algorithm for timing driven logic replication in FPGA. Master's thesis, University of Illinois at Chicago, 2002.

[17] BERAUDO, G., AND LILLIS, J. Timing optimization of FPGA placements by logic replication. In *Proceedings of DAC* (2003), pp. 196–201.

[18] BERNARD, M., AND JACQUENET, F. Free space modeling for placing rectangles without overlapping. *J. of Universal Comp. Sci. 3*, 6 (1997), 703–720.

[19] BETZ, V., AND ROSE, J. VPR: A new packing, placement and routing tool for FPGA research. In *Field-Programmable Logic and Applications* (1997), W. Luk, P. Y. Cheung, and M. Glesner, Eds., Springer-Verlag, Berlin, pp. 213–222.

[20] BRENNER, U., AND VYGEN, J. Legalizing a placement with minimum total movement. *IEEE Transactions on Computer-Aided Design 23* (2004), 1597–1613.

[21] CALDWELL, A. E., KAHNG, A. B., KENNINGS, A. A., AND MARKOV, I. L. Hypergraph partitioning for VLSI CAD: methodology for heuristic development, experimentation and reporting. In *Proceedings of DAC* (1999), pp. 349–354.

[22] CALDWELL, A. E., KAHNG, A. B., AND MARKOV, I. L. Design and implementation of the Fiduccia-Mattheyses heuristic for VLSI netlist partitioning. In *Proceedings of Workshop on Algorithm Engineering and Experimentation* (1999), pp. 177–193.

[23] CALDWELL, A. E., KAHNG, A. B., AND MARKOV, I. L. Optimal partitioners and end-case placers for standard-cell layout. In *Proceedings of ISPD* (1999), pp. 90–96.

[24] CALDWELL, A. E., KAHNG, A. B., AND MARKOV, I. L. Can recursive bisection alone produce routable placements? In *Proceedings of DAC* (2000), pp. 477–482.

[25] CHAN, P. K., AND SCHLAG, M. D. F. Parallel placement for field-programmable gate arrays. In *Proceedings of FPGA* (2003), pp. 43–50.

[26] CHAN, T., CONG, J., AND SZE, K. Multilevel generalized force-directed method for circuit placement. In *Proceedings of ISPD* (2005), pp. 185–192.

[27] CHAN, T. F., CONG, J., SHINNERL, J. R., SZE, K. S., AND XIE, M. mPL6: Enhanced multilevel mixed-size placement. In *Proceedings of ISPD* (2006), pp. 212–214.

[28] CHANG, C.-C., CONG, J., ROMESIS, M., AND XIE, M. Optimality and scalability study of existing placement algorithms. *Transactions on DISC 23*, 4 (April 2004), 537–549.

[29] CHAUDHARY, K., AND NAG, S. K. Method for analytical placement of cells using density surface representations. United States Patent 6,415,425, July 2002.

[30] CHEN, D. T., VORWERK, K., AND KENNINGS, A. Improving timing-driven FPGA packing with physical information. In *Proceedings of FPL* (2007), pp. 117–123. Nominated for Best Paper Award.

[31] CHEN, G., AND CONG, J. Simultaneous timing driven clustering and placement for FPGAs. In *Proceedings of FPL* (2004), pp. 158–167.

[32] CHEN, G., AND CONG, J. Simultaneous timing-driven placement and duplication. In *Proceedings of FPGA* (2005), pp. 51–59.

[33] CHEN, T.-C., JIANG, Z.-W., HSU, T.-C., CHEN, H.-C., AND CHANG, Y.-W. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design 27*, 7 (July 2008), 1228–1240.

[34] CHEN, T.-C. C., HSU, T.-C., JIANG, Z.-W., AND CHANG, Y.-W. NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs. In *Proceedings of ISPD* (2005), pp. 236–238.

[35] CHENG, C.-K., AND KUH, E. Module placement based on resistive network optimization. *IEEE Transactions on Computer-Aided Design 3*, 3 (July 1984), 218–225.

[36] CHERKASSKY, B. V., AND GOLDBERG, A. V. On implementing the push-relabel method for the maximum flow problem. *Algorithmica 19*, 4 (1997), 390–410.

[37] COIN-OR FOUNDATION. Computational infrastructure for operations research. `http://www.coin-or.org`, 2006.

[38] CONG, J., KONG, T., SHINNERL, J. R., XIE, M., AND YUAN, X. Large-scale circuit placement: Gap and promise. In *Proceedings of ICCAD* (2003), pp. 883–890.

[39] CONG, J., AND ROMESIS, M. Performance-driven multi-level clustering with application to hierarchical FPGA mapping. In *Proceedings of DAC* (2001), pp. 389–394.

[40] CONG, J., AND XIE, M. A robust detailed placement for mixed-size IC designs. In *Proceedings of ASPDAC* (2006), pp. 188–194.

[41] CORP., A. *HardCopy Series Handbook, Volume 1: Section 1: HardCopy II Device Family Data Sheet*. Altera, 2005.

[42] DE MICHELI, G. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

[43] DEHKORDI, M., AND BROWN, S. Performance-driven recursive multi-level clustering. In *Proceedings of FPL* (2003), pp. 262–269.

[44] DOLL, K., JOHANNES, F. M., AND ANTREICH, K. J. Iterative placement improvement by network flow methods. *IEEE Transactions on Computer-Aided Design 13*, 10 (October 1994), 1189–1200.

[45] DUNLOP, A. E., AND KERNIGHAN, B. W. A placement procedure for standard-cell VLSI circuits. *IEEE Transactions on Computer-Aided Design 4*, 1 (January 1985), 92–98.

[46] EISENMANN, H., AND JOHANNES, F. M. Generic global placement and floorplanning. In *Proceedings of DAC* (1998), ACM Press, pp. 269–274.

[47] ETAWIL, H., AREIBI, S., AND VANNELLI, A. Attractor-repeller approach for global placement. In *Proceedings of ICCAD* (1999), pp. 20–24.

[48] ETAWIL, H. A. Y. *Convex Optimization and Utility Theory: New Trends in VLSI Circuit Layout*. Ph. D. thesis, University of Waterloo, 1999.

[49] FIDUCCIA, C. M., AND MATTHEYSES, R. M. A linear-time heuristic for improving network partitions. In *Proceedings of DAC* (1982), pp. 175–181.

[50] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[51] GOLDBERG, A. V., AND TARJAN, R. E. A new approach to the maximum-flow problem. *Journal of the ACM 35*, 4 (1988), 921–940.

[52] GOTO, S. An efficient algorithm for the two-dimensional placement problem in electrical circuit layout. *IEEE Transactions on Computer Aided Systems CAS-28*, 1 (1981), 12–18.

[53] HAHN, P. M., HIGHTOWER, W. L., JOHNSON, T. A., GUIGNARD-SPIELBERG, M., AND ROUCAIROL, C. Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem. *Yugoslav Journal of Operations Research 11*, 1 (2001), 41–60.

[54] HALL, K. M. An *r*-dimensional quadratic placement algorithm. *Management Science 17* (November 1970), 219–229.

[55] HENTSCHKE, R. F., AND AUGUSTO DA LUZ REIS, R. Improving simulated annealing placement by applying random and greedy mixed perturbations. In *Proceedings of Integrated Circuits and Systems Design* (Washington, 2003), IEEE, pp. 267–272.

[56] HILL, D. Method and system for high speed detailed placement of cells within an integrated circuit design. United States Patent 6,370,673, April 2002.

[57] HRKIC, M., LILLIS, J., AND BERAUDO, G. An approach to placement-coupled logic replication. In *Proceedings of DAC* (2004), pp. 711–716.

[58] HU, B., AND MAREK-SADOWSKA, M. Congestion minimization during placement without estimation. In *Proceedings of ICCAD* (2002), pp. 739–745.

[59] HU, B., AND MAREK-SADOWSKA, M. Multilevel expansion-based VLSI placement with blockages. In *Proceedings of ICCAD* (November 2004), pp. 558–564.

[60] HU, B., ZENG, Y., AND MAREK-SADOWSKA, M. mFAR: Fixed-points-addition-based VLSI placement algorithm. In *Proceedings of ISPD* (April 2005), pp. 239–241.

[61] HUR, S.-W., AND LILLIS, J. Mongrel: Hybrid techniques for standard cell placement. In *Proceedings of ICCAD* (2000), IEEE Press, pp. 165–170.

[62] HUSTIN, S., AND SANGIOVANNI-VINCENTELLI, A. TIM, a new standard cell placement program based on the simulated annealing algorithm. In *IEEE Physical Design Workshop* (1987).

[63] KAHNG, A. B., AND REDA, S. Placement feedback: A concept and method for better min-cut placements. In *Proceedings of DAC* (2004), pp. 357–362.

[64] KAHNG, A. B., REDA, S., AND WANG, Q. Aplace: A general analytic placement framework. In *Proceedings of ISPD* (2005), pp. 233–235.

[65] KAHNG, A. B., AND WANG, Q. An analytic placer for mixed-size placement and timing-driven placement. In *Proceedings of ICCAD* (November 2004), pp. 565–572.

[66] KAHNG, A. B., AND WANG, Q. Implementation and extensibility of an analytic placer. In *Proceedings of ISPD* (2004), pp. 18–25.

[67] KARYPIS, G. *Multilevel Optimization and VLSICAD*. Kluwer Academic Publishers, Boston, 2002, ch. 3.

[68] KARYPIS, G., AGGARWAL, R., KUMAR, V., AND SHEKHAR, S. Multilevel hypergraph partitioning: Applications in VLSI domain. *IEEE Transactions on VLSI Syst. 7*, 1 (March 1999), 69–79.

[69] KENNINGS, A., AND MARKOV, I. L. Analytical placement of hypergraphs. Tech. Rep. 990020, UCSD VLSI CAD Laboratory, January 1999.

[70] KENNINGS, A., AND MARKOV, I. L. Analytical minimization of half-perimeter wirelength. In *Proceedings of ASPDAC* (2000), pp. 179–184.

[71] KENNINGS, A., AND VORWERK, K. Force-directed and other continuous placement methods. In *Handbook of Algorithms for Physical Design Automation*, D. Mehta, C. Alpert, and S. Sapatnekar, Eds. CRC Press, 2007.

[72] KERNIGHAN, B. W., AND LIN, S. An efficient heuristic procedure for partitioning graphs. *Bell Sys. Tech. Journal 49*, 2 (1970), 291–308.

[73] KHATKHATE, A., LI, C., AGNIHOTRI, A. R., YILDIZ, M. C., ONO, S., KOH, C.-K., AND MADDEN, P. H. Recursive bisection based mixed block placement. In *Proceedings of ISPD* (2004), pp. 84–89.

[74] KIM, J.-G., AND KIM, Y.-D. A linear programming-based algorithm for floorplanning in VLSI design. *IEEE Transactions on Computer-Aided Design 2*, 5 (2003), 584–592.

[75] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science* (May 1983), 671–680.

[76] KLEINHANS, J. M., SIGL, G., JOHANNES, F. M., AND ANTREICH, K. J. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design 10*, 3 (March 1991), 356–365.

[77] KONG, T. A novel net weighting algorithm for timing-driven placement. In *Proceedings of ICCAD* (2002), pp. 172–176.

[78] LEIGHTON, F. T. *Complexity issues in VLSI: Optimal layouts for the shuffle-exchange graph and other networks*. MIT Press, 1983.

[79] LI, J., LILLIS, J., LIU, L.-T., AND CHENG, C.-K. New spectral linear placement and clustering approach. In *Proceedings of DAC* (1996), pp. 88–93.

[80] LIN, J.-M., AND CHANG, Y.-W. TCG-S: orthogonal coupling of P*-admissible representations for general floorplans. In *Proceedings of DAC* (2002), pp. 842–847.

[81] LUDWIN, A., BETZ, V., AND PADALIA, K. High-quality, deterministic parallel placement for FPGAs on commodity hardware. In *Proceedings of FPGA* (2008), pp. 14–23.

[82] M. WANG, X. Y., AND SARRAFZADEH, M. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proceedings of ICCAD* (November 2000), pp. 260–263.

[83] MAIDEE, P., ABABEI, C., AND BAZARGAN, K. Fast timing-driven partitioning-based placement for island style FPGAs. In *Proceedings of DAC* (2003), ACM Press, pp. 598–603.

[84] MCMURCHIE, L., AND EBELING, C. Pathfinder: A negotiation-based performance-driven router for FPGAs. In *Proceedings of FPGA* (1995), pp. 111–117.

[85] MITRA, D., ROMEO, F., AND SANGIOVANNI-VINCENTELLI, A. Convergence and finite-time behavior of simulated annealing. In *Proceedings of of the Conference on Decision and Control* (1985), pp. 761–767.

[86] MOFFITT, M. D., NG, A. N., MARKOV, I. L., AND POLLACK, M. E. Constraint-driven floorplan repair. In *Proceedings of DAC* (July 2006), pp. 1103–1108.

[87] MOGAKI, M., MIURA, C., AND TERAI, H. Algorithm for block placement with size optimization technique by the linear programming approach. In *Proceedings of ICCAD* (1987), pp. 80–83.

[88] MULPURI, C., AND HAUCK, S. Runtime and quality tradeoffs in FPGA placement and routing. In *Proceedings of FPGA* (2001), pp. 29–36.

[89] MURATA, H., FUJIYOSHI, K., NAKATAKE, S., AND KAJITANI, Y. Rectangle-packing-based module placement. In *Proceedings of ICCAD* (1995), pp. 472–479.

[90] NAG, S., AND CHAUDHARY, K. Post-placement residual-overlap removal with minimal movement. In *Proceedings of DATE* (1999), pp. 581–586.

[91] NAM, G.-J. ISPD 2006 placement contest: Benchmark suite and results. In *Proceedings of ISPD* (2006), pp. 167–167.

[92] NAM, G.-J., ALPERT, C. J., VILLARRUBIA, P., WINTER, B., AND YILDIZ, M. The ISPD2005 placement contest and benchmark suite. In *Proceedings of ISPD* (2005), pp. 216–220.

[93] NAYLOR, W., DONELLY, R., AND SHA, L. Non-linear optimization system and method for wire length and density within an automatic electronic circuit placer. United States Patent 6,662,348, July 2001.

[94] NG, A. N., MARKOV, I. L., AGGARWAL, R., AND RAMACHANDRAN, V. Solving hard instances of floorplacement. In *Proceedings of ISPD* (2006), pp. 170–177.

[95] PAN, M., VISWANATHAN, N., AND CHU, C. An efficient and effective detailed placement algorithm. In *Proceedings of ICCAD* (2005), pp. 48–55.

[96] PERRY, D. *VHDL*. McGraw-Hill, 1998.

[97] RAMACHANDARAN, P., AGNIHOTRI, A. R., ONO, S., DAMODARAN, P., SRIHARI, K., AND MADDEN, P. H. Optimal placement by branch-and-price. In *Proceedings of ASPDAC* (2005), pp. 337–341.

[98] ROY, J. A., PAPA, D. A., ADYA, S. N., CHAN, H. H., NG, A. N., LU, J. F., AND MARKOV, I. L. Capo: Robust and scalable open-source min-cut floorplacer. In *Proceedings of ISPD* (2005), pp. 224–226.

[99] ROY, J. A., PAPA, D. A., AND MARKOV, I. L. Fine control of local whitespace in placement. In *VLSI Design* (2008), vol. 2008. 10 pages.

[100] ROY, J. A., PAPA, D. A., NG, A. N., AND MARKOV, I. L. Satisfying whitespace requirements in top-down placement. In *Proceedings of ISPD* (2006), pp. 206–208.

[101] SECHEN, C., AND SANGIOVANNI-VINCENTELLI, A. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits* (April 1985), 510–522.

[102] SEGURA, E. C. Evolutionary computation with simulated annealing: Conditions for optimal equilibrium distribution. *Journal of Computer Science and Technology 5*, 4 (2005), 178–182.

[103] SHAHOOKAR, K., AND MAZUMDER, P. VLSI cell placement techniques. *ACM Comput. Surv. 23*, 2 (1991), 143–220.

[104] SIGL, G., DOLL, K., AND JOHANNES, F. M. Analytical placement: A linear or a quadratic objective function? In *Proceedings of DAC* (1991), pp. 427–432.

[105] SINGH, A., AND MAREK-SADOWSKA, M. Efficient circuit clustering for area and power reduction in fpgas. In *Proceedings of FPGA* (2002), pp. 59–66.

[106] SINGH, S. P., AND SHARMA, R. R. K. A review of different approaches to the facility layout problems. *Int. J. Adv. Manuf. Technol. 30* (2006), 425–433.

[107] SMITH, M. J. S. *Application-specific integrated circuits*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[108] SPINDLER, P., AND JOHANNES, F. M. *Kraftwerk: A Fast and Robust Quadratic Placer Using an Exact Linear Net Model*. Springer-Verlag, New York, 2007, pp. 59–93.

[109] SUARIS, P. R., AND KEDEM, G. Standard cell placement by quadrisection. In *Proceedings of ICCAD* (1987), pp. 612–615.

[110] SUN, W.-J., AND SECHEN, C. Efficient and effective placement for very large circuits. *IEEE Transactions on Computer-Aided Design 14*, 3 (March 1995), 349–359.

[111] SUTANTHAVIBUL, S., SHRAGOWITZ, E., AND ROSEN, J. B. An analytical approach to floorplan design and optimization. *IEEE Transactions on Computer-Aided Design 10*, 6 (1991), 761–769.

[112] SWARTZ, W., AND SECHEN, C. New algorithms for the placement and routing of macro cells. In *Proceedings of ICCAD* (1990), pp. 336–339.

[113] SWARTZ, W., AND SECHEN, C. Timing driven placement for large standard cell circuits. In *Proceedings of DAC* (1995), pp. 211–215.

[114] SZE, C., WANG, T.-C., AND WANG, L.-C. Multilevel circuit clustering for delay minimization. In *IEEE Transactions on Computer-Aided Design* (2004), pp. 1073–1085.

[115] TAKAMURA, A., AND FUKASAKU, I. TITAC-2: An asynchronous 32-bit microprocessor based on scalable-delay-insensitive model. In *Proceedings of ICCD* (1997), p. 288.

[116] TESSIER, R. Fast placement approaches for FPGAs. *IEEE Transactions on Design Automation of Electronic Systems 7*, 2 (2002), 284–305.

[117] TSAY, R.-S., KUH, E., AND HSU, C.-P. PROUD: A sea-of-gates placement algorithm. *IEEE Design and Test of Computers 5*, 6 (December 1988), 44–56.

[118] ULLMAN, J. *Computational Aspects of VLSI*. Computer Science Press, 1984.

[119] VICENTE, J. D., LANCHARES, J., AND HERMIDA, R. Annealing placement by thermodynamic combinatorial optimization. *IEEE Transactions on Design Automation of Electronic Systems 9*, 3 (2004), 310–332.

[120] VISWANATHAN, N., AND CHU, C. C.-N. Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Proceedings of ISPD* (2004), pp. 26–33.

[121] VORWERK, K. On the engineering of a stable force-directed placer. Master's thesis, University of Waterloo, 2004.

[122] VORWERK, K., ANJOS, M. F., AND KENNINGS, A. VLSI floorplan repair using dynamic whitespace management, constraint graphs and linear programming. *Journal of Engineering Optimization 40*, 6 (2008), 559–577.

[123] VORWERK, K., AND KENNINGS, A. Mixed-size placement via line search. In *Proceedings of ICCAD* (2005), pp. 899–904.

[124] VORWERK, K., AND KENNINGS, A. Robust minimum movement legalization with partitioning, constraint graphs and intelligent white space management. Tech. Rep. UW-ECE-2006-08, University of Waterloo, 2006.

[125] VORWERK, K., KENNINGS, A., CHEN, D. T., AND BEHJAT, L. Floorplan repair using dynamic whitespace management. In *Proceedings of GLSVLSI* (2007), pp. 552–557.

[126] VORWERK, K., KENNINGS, A., GREENE, J., AND CHEN, D. T. Improving annealing via directed moves. In *Proceedings of FPL* (2007), pp. 363–370. Recipient of Best Paper Award.

[127] VORWERK, K., KENNINGS, A., AND GREENE, J. W. Improving simulated annealing-based FPGA placement with directed moves. *IEEE Transactions on Computer-Aided Design 28*, 2 (2009), 179–192.

[128] VORWERK, K., KENNINGS, A., AND VANNELLI, A. Engineering details of a stable analytic placer. In *Proceedings of ICCAD* (November 2004), pp. 573–580.

[129] VYGEN, J. Algorithms for large-scale flat placement. In *Proceedings of DAC* (1997), pp. 746–751.

[130] VYGEN, J. Four-way partitioning of two-dimensional sets. Unpublished Manuscript, 2000.

[131] WANG, M., AND SARRAFZADEH, M. On the behavior of congestion minimization during placement. In *Proceedings of ISPD* (1999), pp. 145–150.

[132] WOOLF, M. B. *Faster Construction Projects with CPM Scheduling*. McGraw Hill, 2007.

[133] WU, K.-C., AND TSAI, Y.-W. Structured ASIC, evolution or revolution? In *Proceedings of ISPD* (2004), pp. 103–106.

[134] YANG, S. Logic synthesis and optimization benchmarks, version 3.0. Tech. rep., Microelectronics Center of North Carolina, 1991.

[135] YANG, X., KASTNER, R., AND SARRAFZADEH, M. Congestion estimation during top-down placement. In *Proceedings of ISPD* (2001), pp. 164–169.

[136] YAO, B., CHEN, H., CHENG, C.-K., CHOU, N.-C., LIU, L.-T., AND SUARIS, P. Unified quadratic programming approach for mixed mode placement. In *Proceedings of ISPD* (2005), pp. 193–199.

[137] ZAHIRI, B. Structured ASICs: opportunities and challenges. In *Proceedings of ICCD* (2003), pp. 404–409.

[138] ZHUO, Y., LI, H., ZHOU, Q., CAI, Y., AND HONG, X. New timing and routability driven placement algorithms for FPGA synthesis. In *Proceedings of GLSVLSI* (2007), pp. 570–575.

Glossary

Nomenclature

In this work, the terms *module*, *cell*, and *node* are used to describe a standard or macrocell. *Macrocell* and *macro block* are also used interchangeably. Similarly, *net*, *wire*, and *interconnect* are used synonymously. The term *pad* is used to refer to the *terminals* of the chip. Moreover, *placement* and *solution* (to the placement problem) are used synonymously to represent an assignment of modules to physical locations on the chip [5]. The term *placer* refers to a tool which implements a heuristic to place cells. *Costing* is used to represent the act of computing the *delta costs* for a move in the simulated annealing strategies described herein.

Terms

Aborted Move	Any move, in the standard cell annealer, which could not be costed. Generally occurs when the move legalizers are unable to retain legality, and the move must be rejected.
ASIC	*Application Specific Integrated Circuit*.
BLE	*Basic Logic Element*. A logic element in an FPGA which typically consists of a FF and LUT.
Bookshelf	An Internet-based repository for literature, benchmarks, and files related to VLSI CAD. Available at: `http://www.gigascale.org/bookshelf`.
Capo	A recursive, minimum-cut bi-partitioning placement tool, available at: `http://vlsicad.eecs.umich.edu/BK/PDtools`.
CAD	*Computer Aided Design*.

CLB	*Combinational Logic Element*. A logic element in an FPGA which typically consists of several BLEs.
CPU	*Central Processing Unit*. Software run-times, in this thesis, are generally reported in terms of the number of minutes spent executing on a CPU.
Dragon	A partitioning- and simulated annealing-based placement tool, available at: `http://er.cs.ucla.edu/Dragon`.
FF	*Flip-flop*.
FPGA	*Field Programmable Gate Array*. An integrated circuit where the logic and wiring of the device can be reprogrammed after its manufacture. An FPGA consists of an array of logic elements (which may include gates and look-up tables), connected by programmable interconnect wiring.
HPWL	*Half Perimeter Wire Length*. An approximation to the actual wire length required to route a design. HPWL is calculated based on the lengths of the horizontal and vertical spans of all nets.
IC	*Integrated Circuit*.
I/O	*Input/Output*.
Kraftwerk	A commercial force-directed placer based on [1].
LUT	*Look-up Table*. A logic element in an FPGA which can implement any $k-$input function.
Macrocell	A cell whose dimensions are neither defined nor constrained by the standard cell library.
Mixed-Size Design	A circuit which includes a mix of both standard cells and macrocells.
NP	*Non-deterministic Polynomial*. A set of computational decision problems which are solvable by a non-deterministic Turing Machine in a number of steps that is a polynomial function of the size of the input. An exponential amount of time may be required to discover the solution, but a potential solution must be verifiable in polynomial time.
QAP	*Quadratic Assignment Problem*. Described in [3, 4].
QOR	*Quality of Result*. A term typically used in reference to the value of the cost function employed during placement.
RAM	*Random Access Memory*.

Standard Cell	A cell whose dimensions are specified in a standard library.
Stochastic Process	A stochastic process is a collection of interdependent random variables which arise when a system's subsequent state is determined both by the process's predictable actions and by a random element [2].
VLSI	*Very Large Scale Integration*.
VPR	*Versatile Place and Route*. A placement and routing tool for FPGA research, available at: `http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html`.
WL	*Wire Length*.

Bibliography

[1] EISENMANN, H., AND JOHANNES, F. M. Generic global placement and floorplanning. In *Proceedings of DAC* (1998), ACM Press, pp. 269–274.

[2] GRAY, R., AND SHIELDS, P. *Probability, random processes, and ergodic properties*. Springer-Verlag Berlin, 1988.

[3] HAHN, P. M., HIGHTOWER, W. L., JOHNSON, T. A., GUIGNARD-SPIELBERG, M., AND ROUCAIROL, C. Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem. *Yugoslav Journal of Operations Research 11*, 1 (2001), 41–60.

[4] HALL, K. M. An *r*-dimensional quadratic placement algorithm. *Management Science 17* (November 1970), 219–229.

[5] SHAHOOKAR, K., AND MAZUMDER, P. VLSI cell placement techniques. *ACM Comput. Surv. 23*, 2 (1991), 143–220.

APPENDIX A

Related Papers

The following works were published as a result of the work described in, related to, or written in conjunction with this thesis.

[1] VORWERK, K., ANJOS, M. F., AND KENNINGS, A. VLSI floorplan repair using dynamic whitespace management, constraint graphs and linear programming. *Journal of Engineering Optimization 40*, 6 (2008), 559–577.

[2] VORWERK, K., AND KENNINGS, A. Robust minimum movement legalization with partitioning, constraint graphs and intelligent white space management. Tech. Rep. UW-ECE-2006-08, University of Waterloo, 2006.

[3] VORWERK, K., KENNINGS, A., CHEN, D. T., AND BEHJAT, L. Floorplan repair using dynamic whitespace management. In *Proceedings of GLSVLSI* (2007), pp. 552–557.

[4] VORWERK, K., KENNINGS, A., GREENE, J., AND CHEN, D. T. Improving annealing via directed moves. In *Proceedings of FPL* (2007), pp. 363–370. Recipient of Best Paper Award.

[5] VORWERK, K., KENNINGS, A., AND GREENE, J. W. Improving simulated annealing-based FPGA placement with directed moves. *IEEE Transactions on Computer-Aided Design 28*, 2 (2009), 179–192.

APPENDIX B

On the Statistical Variability of Stochastic Placement Techniques

Randomness is a key attribute which works both *for* and *against* stochastic optimization methods. By changing the value of the initial random seed, a simulated annealing-based placer can produce widely-varying results.

To ascertain just how variable an annealing-based placement strategy can be, the well-known placer, `VPR` [2], was tested using multiple initial random seeds. Specifically, each design in the MCNC [4] suite was placed and routed 50 times using different starting seed values, and the post-routed wire lengths and critical path delays were measured. An architecture comprising 1 BLE/CLB with low-stress routing was employed for these tests.

The results for routed wire length and critical path delays are presented in Tables B.1 and B.2. These tables present the ratios of the minimum-to-average value, maximum-to-average value, maximum-to-minimum, and the result from an Anderson-Darling test for normality [1] which indicates the confidence level that the data are normal.

In terms of the wire length results, approximately 7 of the 20 designs could be considered normal, 4 of the designs are strongly not-normal, and the remainder are indeterminate. The range of results (as measured by the ratio of the maximum to the minimum wire lengths for each design) is reasonably small, with most designs offering less than 10% swing depending on the random seed. In most cases, the averages are reasonably well-centred with respect to the spread of the results.

The critical path delay values, however, are quite different. The Anderson-Darling normality test suggests that only 1 of the 20 designs is strongly normal, 12 are not-normal, and the remainder

Table B.1: Summary of post-routed wire length variability in VPR.

Design	Min/Avg	Max/Avg	Max/Min	Normality Confidence
alu4	0.96	1.04	1.08	48.18%
apex2	0.94	1.03	1.1	5.32%
apex4	0.97	1.03	1.07	68.00%
bigkey	0.94	1.1	1.17	13.89%
clma	0.96	1.06	1.1	87.03%
des	0.95	1.07	1.13	82.19%
diffeq	0.97	1.04	1.08	33.95%
dsip	0.93	1.1	1.19	18.63%
elliptic	0.95	1.04	1.09	97.49%
ex1010	0.98	1.02	1.04	31.31%
ex5p	0.97	1.04	1.07	77.32%
frisc	0.97	1.03	1.06	32.85%
misex3	0.95	1.04	1.09	89.81%
pdc	0.97	1.03	1.06	31.72%
s298	0.97	1.03	1.07	63.94%
s38417	0.98	1.03	1.06	2.94%
s38584.1	0.98	1.02	1.05	45.57%
seq	0.97	1.04	1.07	82.19%
spla	0.98	1.02	1.04	83.53%
tseng	0.96	1.03	1.07	52.65%

are indeterminate. The kurtosis for these data was measured, and it suggests that the variance is due to infrequent extreme deviations, as opposed to frequent modestly-sized deviations. In other words, critical path delays can often be affected by very large outliers.

Furthermore, the skewness of these data was computed, and it suggests that most designs follow a positively-skewed pattern. (In other words, the shape of the data looks somewhat like a normal curve whose "bell" has been shifted to the left.) This tends to agree with the observation that the minimum-to-average ratio is smaller than the maximum-to-average ratio—in other words, the average of the critical path delays tends to be closer to the minimums than to the maximums. It is worth noting that the ratio of maximum-to-minimum critical path delay is quite large—often on the order of ≈ 1.3, and as high as 1.91.

One interesting question which arises from these statistics is: how have results been computed,

Table B.2: Summary of post-routed critical path variability in VPR.

Design	Min/Avg	Max/Avg	Max/Min	Normality Confidence
alu4	0.89	1.14	1.28	17.06%
apex2	0.95	1.1	1.16	5.79%
apex4	0.93	1.2	1.28	0.09%
bigkey	0.83	1.59	1.91	0.30%
clma	0.77	1.15	1.49	9.39%
des	0.88	1.17	1.33	40.31%
diffeq	0.87	1.14	1.3	50.35%
dsip	0.89	1.22	1.37	1.62%
elliptic	0.88	1.15	1.3	3.35%
ex1010	0.9	1.09	1.2	84.90%
ex5p	0.91	1.16	1.27	0.00%
frisc	0.81	1.12	1.37	0.01%
misex3	0.89	1.17	1.31	4.06%
pdc	0.85	1.15	1.35	18.30%
s298	0.95	1.06	1.11	30.73%
s38417	0.89	1.18	1.32	41.77%
s38584.1	0.87	1.16	1.33	9.80%
seq	0.9	1.2	1.33	0.30%
spla	0.93	1.18	1.27	0.00%
tseng	0.95	1.06	1.13	29.17%

historically, in the academic literature? This question is difficult to answer, as there are no known papers (to the author's knowledge) which have examined the question of reliable, statistical reporting of results for VLSI CAD. It would not be unsurprising to discover that, in their quest for improvement, many authors have run multiple seeds, only to pick the most favourable results for their papers.

At the same time, deriving a rigorous, statistical methodology for comparing results in the CAD literature is difficult and potentially error-prone. While an analysis of variance could no doubt strengthen one's confidence in comparing two data sets, there are several practical considerations to make: improperly used, such statistics can easily lead one to draw incorrect conclusions, and the time required to establish such a level of rigour may be impractical. How, then, can one be reasonably assured that results presented in a paper are reasonable?

The approach taken in this thesis is to minimize the statistical variability of the results by employing multiple random seeds for each design-under-test and *averaging* the results across the seeds. (That is to say, if a circuit were to be placed for a specific architecture under specific conditions, rather than placing it once, it would be placed several times, and the reported result would be the average of all seeds.) In this way, the chance of coming across a serious outlier is minimized.

Establishing *how many* seeds are necessary is a matter of trading-off confidence in the results with the practical consideration of run-time. In the case of wire length-driven optimization, the empirical observations presented here suggest that fewer seeds may be necessary (since there is less variance in the results), whereas timing-driven optimizations may require more samples to minimize variability.

To derive an approximate number of the minimum number of seeds required to achieve sufficiently high confidence in the results, a statistical power computation was performed using `G*Power` [3]. The wire length and timing results from `VPR` were input into the tool and a maximum false negative rate of $\beta = 0.20$ was employed. (The false negative rate is the probability of making a Type II statistical error—or, stated differently, the error of failing to observe a difference when, in fact, there is one.) `G*Power` estimated that, on average, 3 to 5 seeds were required per design to achieve the desired statistical confidence.

Thus, unless noted otherwise, the placement tools implemented in this thesis were run with a minimum of 3 seeds when used in wire length-driven mode, and a minimum of 5 seeds when used in timing-driven mode. It is hoped that by averaging the results rather than taking the extreme outliers, more meaningful comparisons can be drawn.

Bibliography

[1] ANDERSON, T. W., AND DARLING, D. A. Asymptotic theory of certain "goodness-of-fit" criteria based on stochastic processes. *Annals of Mathematical Statistics 23* (1952), 193–212.

[2] BETZ, V., AND ROSE, J. VPR: A new packing, placement and routing tool for FPGA research. In *Field-Programmable Logic and Applications* (1997), W. Luk, P. Y. Cheung, and M. Glesner, Eds., Springer-Verlag, Berlin, pp. 213–222.

[3] ERDFELDER, E., FAUL, F., AND BUCHNER, A. GPOWER: A general power analysis program. *Behavior Research Methods, Instruments, and Computers 28* (1996), 1–11.

[4] YANG, S. Logic synthesis and optimization benchmarks, version 3.0. Tech. rep., Microelectronics Center of North Carolina, 1991.

APPENDIX C

Implementation Details and Data Structures

This chapter discusses some of the key data structures used to represent the architectures and netlists in the annealers described in this thesis. The FPGA tools and standard cell tools each possess very different requirements, not just in terms of device modelling, but also in terms of the size of designs that must be processed.

FPGA Tools

The flow for the placement tool described in Chapter 3 is similar, but not identical, to `VPR` [3]. Several salient differences exist between the two.

Rather than employing a separate clustering and placement tool, `KPF` incorporates both clustering and placement in one. It should be noted that, for the clustering results examined in Chapter 3, all clusterings were produced by `T-VPack` [3], even though `KPF` includes the `HDPack` [5] clustering tool.

Modelling FPGA Devices

The circuit and device data structures are also quite different from `VPR`. The device data for `KPF` are specified in `Python` [1], and are read-in by a `Python` interpreter embedded inside `KPF`. During compile time, the program `Swig` [2] automatically creates a C++-to-`Python` API which allows a user-defined set of C++ functions to be called from within `Python`, so that the embedded interpreter can call back into C++ routines. This allows the `Python` interpreter to manipulate C++ data structures.

The architecture specification for `KPF` is very flexible, and allows the user to specify archi-

tectural dimensions, locations of all BLEs, CLBs, and RAMs, hierarchical relationships between the LUTs, FFs, BLEs, and CLBs, heterogeneous layouts, as well as a flexible set of clustering constraints. Clustering constraints can be specified in terms of limits on the number of control signals, limits on the number of data signals fed back within a CLB, limits on outputs, limits on arbitrary combinations of signals (e.g., a user can specify that the number of clock and enable signals be less than or equal to 3, but the total number of signals be less than 32), and limits on the number of logic modules within both BLEs and CLBs. Most of these features were not described in this thesis, as comparisons were made primarily to `VPR`-style architectures, although they were built into `KPF` in anticipation of future extensibility.

For this thesis, a separate executable was created to automatically generate the `Python` architecture files for each design in the MCNC [7] suite; the reason for this was primarily to facilitate comparison with `VPR`'s ability to "auto-size" the devices for each design. For realistic FPGA device modelling, where only one or two die sizes may be tested, it is more likely that a user would simply hand-code the architecture file, or perhaps write a program to convert the IC layout automatically to the desired format.

In `KPF`, the FPGA "grid" is modelled in a hierarchical fashion. A "layer" holds a matrix of "sites". Perhaps unique to this tool is the fact that the physical locations of sites do not *have* to align with their position in the matrix, though, in practice, this is a good idea as it makes it much faster to convert between physical coordinates and indices in the placement grid. (This is especially useful when using median improvement moves [6], as one must be able to quickly convert the physical locations into grid coordinates.)

In `KPF`, there are multiple layers for each type of cell, such as a FF, LUT, BLE, and so forth. This is just one way to model a FPGA architecture; an alternative technique would have been to simply place all cells within one (large) placement grid. Both approaches are viable, but trade-off the complexity of iterating over available grid sites (which is the greatest advantage of the multi-layer approach) with the ability to quickly convert physical locations to matrix indices (which is the greatest advantage of a single-grid approach).

Modelling FPGA Netlists

The netlists in `KPF` use a node-pin-edge format, where the nodes represent placeable cells, the edges represent the nets in the design, and the pins represent the connections between nodes and edges. The types of pins and nodes are pre-defined, with pins characterized by their `PinDescription`s and nodes defined by their `CellDescription`s. These pin and cell descriptions coincide with the information supplied by the `Python` architecture file (which can configure the various properties, such as pin offsets and types).

Timing, in KPF, is done entirely on the netlist hypergraph, as it was felt that this offered the greatest flexibility. For instance, in the event that the netlist were to be changed, a separate timing graph data structure would not have to be recreated.

Because KPF supports pin offsets, careful consideration had to be made when designing the incremental bounding-box algorithm, which is conceptually based on [3]. Only the larger cells possess pin offsets (and not, generally, the core logic), as the pin offsets incur computational overhead and increase the likelihood of cache misses when computing bounding box caches.

Example of a Python-based FPGA Architecture

Consider the following architecture file for VPR [3], which defines a 4 BLE/CLB architecture:

```
io_rat 4
inpin class: 0 bottom
inpin class: 0 left
inpin class: 0 top
inpin class: 0 right
inpin class: 0 bottom
inpin class: 0 left
inpin class: 0 top
inpin class: 0 right
inpin class: 0 bottom
inpin class: 0 left
outpin class: 1 top
outpin class: 1 right
outpin class: 1 bottom
outpin class: 1 left
inpin class: 2 global top
subblocks_per_clb 4
subblock_lut_size 4
```

Similar functionality is achievable using the Python scripting language coded in KPF; however, substantially more information must be provided. In fact, the Python architecture file must be specified for every device size to be tested—thus, during testing in this thesis, a separate architecture file was generated for each of the MCNC [7] circuits, for each of the 1, 2, 4, or 8 BLE/CLB architectures under consideration.

The following is a simplified example of an architecture data file format used by KPF to model the 4 BLE/CLB architecture used by the MCNC design alu4. Here, it can be seen how the features of the Python programming language are employed for much of the repetitive processing—the interpreter takes care of the parsing and syntax-checking—and how the Python-to-C++ API embedded inside KPF handles the allocation and construction of the database objects.

Arch File: General Header

```
num_clbs_x          =     20;
num_clbs_y          =     20;
num_bles_per_clb    =      4;
num_iob_rat         =      4;
width_clb           = 1.0000;
height_clb          = 1.0000;
width_ble           = 1.0000;
height_ble          = 1.0000 / num_bles_per_clb;
width_generic_ram   = 1.5000 * width_clb;
height_generic_ram  = 1.5000 * height_clb;
width_lut           = width_ble / 2.0;
height_lut          = height_ble;
```

```
width_dff           = width_ble / 2.0;
height_dff          = height_ble;
width_iob           = 1.0000 / num_iob_rat;
height_iob          = 1.0000;

# Add the device to get started, and set up general properties.
AddDevice();
gp = GlobalProperty();
gp.SetDeviceName( "VPR" );
gp.SetNumBLEsPerCLB( num_bles_per_clb );
```

Arch File: Defining an FF

The following snippet illustrates how a FF with 1 input, 1 output, and 1 clock is defined.

```
cd = ArchCellDescriptor( ArchCellType_FF );
cd.SetHeight( height_dff );
cd.SetWidth( width_dff );
pd = ArchPinDescriptor();
pd.SetId( 0 );
pd.SetType( ArchPinType_DataInput );
pd.SetOffsetX( 0.0 );
pd.SetOffsetY( 0.0 );
cd.AddPin( pd );
pd = ArchPinDescriptor();
pd.SetId( 1 );
pd.SetType( ArchPinType_DataOutput );
pd.SetOffsetX( 0.0 );
pd.SetOffsetY( 0.0 );
cd.AddPin( pd );
pd = ArchPinDescriptor();
pd.SetId( 2 );
pd.SetType( ArchPinType_Clock );
pd.SetOffsetX( 0.0 );
pd.SetOffsetY( 0.0 );
cd.AddPin( pd );
```

Arch File: Defining a LUT

The following snippet illustrates how a LUT with 4 inputs and 1 output is defined.

```
cd = ArchCellDescriptor( ArchCellType_LUT );
cd.SetHeight( height_lut );
cd.SetWidth( width_lut );
pd = ArchPinDescriptor();
pd.SetId( 0 );
pd.SetType( ArchPinType_DataInput );
pd.SetOffsetX( 0.0 );
pd.SetOffsetY( 0.0 );
cd.AddPin( pd );
pd = ArchPinDescriptor();
pd.SetId( 1 );
pd.SetType( ArchPinType_DataInput );
pd.SetOffsetX( 0.0 );
pd.SetOffsetY( 0.0 );
cd.AddPin( pd );
pd = ArchPinDescriptor();
pd.SetId( 2 );
pd.SetType( ArchPinType_DataInput );
pd.SetOffsetX( 0.0 );
pd.SetOffsetY( 0.0 );
cd.AddPin( pd );
pd = ArchPinDescriptor();
pd.SetId( 3 );
pd.SetType( ArchPinType_DataInput );
pd.SetOffsetX( 0.0 );
pd.SetOffsetY( 0.0 );
cd.AddPin( pd );
pd = ArchPinDescriptor();
pd.SetId( 4 );
pd.SetType( ArchPinType_DataOutput );
pd.SetOffsetX( 0.0 );
pd.SetOffsetY( 0.0 );
cd.AddPin( pd );
```

The definitions for BLEs and CLBs follow similarly (but have been omitted for brevity).

Arch File: Creating the Layers

The dimensions of each layer are specified as follows.

```
layer_iob = ArchLayer( ArchCellType_IOB );
layer_clb = ArchLayer( ArchCellType_CLB );
layer_ble = ArchLayer( ArchCellType_BLE );
layer_lut = ArchLayer( ArchCellType_LUT );
layer_dff = ArchLayer( ArchCellType_FF  );
layer_iob.CreateGrid( 22 , 22 );
layer_clb.CreateGrid( 22 , 22 );
layer_ble.CreateGrid( 22 , 22 * num_bles_per_clb );
layer_lut.CreateGrid( 22 , 22 * num_bles_per_clb );
layer_dff.CreateGrid( 22 , 22 * num_bles_per_clb );
```

Arch File: Setting Up the Layers

Each site in each layer is looped over, and the various properties—dimensions, capacity, locations, and parent/children relationships with sites in other layers are thoroughly specified. This portion of the architecture file can be quite long, so only a small sampling is provided below (for brevity).

In this example, note how the `Python` language simplifies the representation and configuration of the architecture. It is worth emphasizing that each of the `Python` objects in this example corresponds to an allocated object in `KPF`'s C++ database.

```
site_clb = layer_clb.GetSite( 0 , 1 );
site_clb.SetXPhys( 0.0000 );
site_clb.SetYPhys( 1.0000 );
site_clb.SetCapacity( 0 );
for subloc in range ( 0 , num_bles_per_clb ):
     site_ble = layer_ble.GetSite( 0 , 1 * num_bles_per_clb + subloc );
     site_ble.SetCapacity( 0 );
     site_ble.SetXPhys( 0.0000 );
     site_ble.SetYPhys( 1.0000 + (subloc * height_ble) );
     site_ble.SetParent( site_clb );
     site_clb.AddChild( site_ble );
     site_lut = layer_lut.GetSite( 0 , 1 * num_bles_per_clb + subloc );
     site_lut.SetCapacity( 0 );
     site_lut.SetXPhys( 0.0000 );
     site_lut.SetYPhys( 1.0000 + (subloc * height_ble) );
     site_lut.SetParent( site_ble );
     site_ble.AddChild( site_lut );
     site_dff = layer_dff.GetSite( 0 , 1 * num_bles_per_clb + subloc );
     site_dff.SetCapacity( 0 );
     site_dff.SetXPhys( 0.0000 + width_lut );
     site_dff.SetYPhys( 1.0000 + (subloc * height_ble) );
     site_dff.SetParent( site_ble );
     site_ble.AddChild( site_dff );
```

Main Data Structures used in the FPGA Tools

In the following code examples, only member variables are shown in order to simplify the presentation.

Architecture Database: Pin Description

This structure defines the basic pin attributes in the architecture, and is configured largely through the `Python` file. (For instance, an architecture site can contain multiple `PinDescriptions`.) There is a predefined set of pin types, but most other properties are configurable.

```
class PinDescription
{
public:
        enum Type {
                Unknown         = 0,
                DataInput       = 1,
                DataOutput      = 2,
                CarryInput      = 3,
                CarryOutput     = 4,
                Enable          = 5,
                Clock           = 6,
                SyncLoad        = 7,
                SyncData        = 8,
                AsyncClear      = 9,
                AsyncSetReset   = 10,
                AsyncLoad       = 11,
                AsyncData       = 12,
                AsyncPre        = 13,
                Pad             = 14,
                _END_TYPE       = 15
        };

protected:
        real64          _offsetX;
        real64          _offsetY;
        uint32          _id;
        Type            _type   : 4;
};
```

Architecture Database: Cell Description

The `CellDescription` models the type of cell. There are a predefined set of cell types, but most other properties are configurable through the `Python` files. Note that the `CellDescription` contains `PinDescription`s, as well as a handful of other ancillary classes (not shown) which largely describe the functionality of the cell in the architecture. For example, the same type of cell can contain multiple different sets of `Configuration`s—this models the fact that some cells may implement slightly different functionality depending upon where they are in the die, but that they are still fundamentally the same type.

```
class CellDescription
{
public:
        enum Type {
                Unknown         = 0,
                BLE             = 1,
                CLB             = 2,
                FF              = 3,
                IOB             = 4,
                LUT             = 5,
                RAM             = 6,
                RAM_SMALL       = 7,
                RAM_MEDIUM      = 8,
                RAM_LARGE       = 9,
                MAC             = 10,
                PLL             = 11,
                JTAG            = 12,
                _END_TYPE       = 13
        };

protected:
        std::vector<Configuration *>            _configurations;        // The configurations for this cell.
        std::vector< std::vector<uint32> >      _pinCache;              // Keeps a list (in vector form) of the types of pins on this node.
        std::vector<SignalConstraint>           _constraints;           // A list of the signal constraints for this cell type.
        std::vector<PinDescription *>           _pinDescriptions;       // A list of the actual pin descriptions.
        DelayDescription                        _dd;                    // Delay information.

        real64                                  _height;
        real64                                  _width;
        uint32                                  _maxNumFeedback;
        uint32                                  _numFeedbackPerCell;
        Type                                    _type                   : 4;
        bool                                    _hasFeedbackConstraint  : 1;
};
```

Architecture Database: Site

This data structure defines an individual, placeable location within the device grid. Note that `Site`s exist inside `Layer`s, and can form hierarchical relationships—for instance, FF and LUT `Site`s can be configured as the children of BLE `Site`s, which are the children of CLB `Site`s. This kind of relationship can be useful for defining complex clustering constraints or when placing carry chain logic.

```
class Architecture::Site
{
public:
        typedef std::vector< Node * >                   Nodes_t;
        typedef std::vector< Architecture::Site * >     Children_t;

private:
        Nodes_t                 _nodes;         // The cells contained in this site.
        Children_t              *_children;     // This site may (or may not) have children.
        Architecture::Layer     *_layer;        // The layer to which this site belongs.
        Architecture::Site      *_parent;       // This site may (or may not) have a parent.
        PointSearch::Location   _physLoc;       // Physical location.
        uint32                  _capacity;      // Number of pieces of logic that can be stored at this location.
        sint32                  _id;            // Unique identifier given to every architectural site.
        uint32                  _occupancy;     // The number of logic pieces that are already at this location.
        sint32                  _subBlock;      // The position in the children's vector in the parent site
                                                // (or -1 for sites without a parent/child relationship).
        uint32                  _xidx;          // The logical X indenx of this site (bottom left corner).
        uint32                  _yidx;          // The logical Y index of this site (bottom left corner).
};
```

Architecture Database: Layer

This class contains the matrix of `Site`s—the placement grid, in effect. A separate `Layer` exists for each type of `CellDescription` in the device.

```
class Architecture::Layer
{
protected:
        std::vector< std::vector< Architecture::Site > >        _grid;

        PointSearch                             _ps;            // Useful for doing fast searches of neighbouring sites
        std::vector< Architecture::Site * >     _sitesByID;     // Stores sites (in the grid) by ID; useful for lookups
        real64                                  _layerXmin;     // Layer dimension
        real64                                  _layerXmax;
        real64                                  _layerYmin;
        real64                                  _layerYmax;
        uint32                                  _nx;            // Max dimension of the layer
        uint32                                  _ny;            // Max dimension of the layer
        CellDescription::Type                   _type;          // Type of this layer
};
```

Netlist: Node

The `Node` class represents a basic, placeable logic unit, and corresponds to a specific type of `CellDescription`, as specified in the device database. `Node`s are placed in `Site` locations. `Node`s are connected to nets (or, `Edge`s) through `Pin`s.

```
class Node
{
public:
        typedef std::vector<Pin *> Pins_t;

protected:
        Pins_t                  _pins;          // All connected pins.
        char *                   _name;          // Name of node.
        Architecture::Site      *_site;         // Location of the node.
        CellDescription         *_cd;           // Type of the cell (its "cell description").
```

```
        sint32                  _id;            // Identifiers.
        sint32                  _config : 8;    // The configuration of the cell;  -1 (error), ... 127.
        FixedMode               _fixed  : 2;    // Status of whether node is fixed.
        PlacedType              _placed : 1;    // How this particular cell has been placed.
};
```

Netlist: Edge

Edges model the nets in the design.

```
class Edge
{
public:
        enum Type {
                Local   = 0,
                Global  = 1
        };

        typedef std::vector<Pin *> Pins_t;

protected:
        Pins_t          _pins;          // Connected pins.
        char *          _name;          // Name of pin.
        sint32          _id;            // Identifier for this pin.
        Type            _type : 1;      // The type of this edge
};
```

Netlist: Pin

Pins act as the connector between `Node`s and `Edge`s. `Pin`s are of pre-defined types, based on their `PinDescription`.

```
class Pin
{
protected:
        Edge *          _edge;          // Edge associated with this pin.
        Node *          _node;          // Node associated with this pin.
        PinDescription *_desc;          // Cached copy of this pin's PinDescription.
        sint32          _idEdge;        // Position in the pins vector for the edge.
        sint32          _idGlobal;      // Global pin ID, unique to every pin in a netlist.  Useful for global lookups.
        sint32          _idNode;        // Position in the pins vector for the node.
};
```

Standard Cell Tools

The modelling of standard cell problems for legalization and detailed placement is similar to the "layer"-based philosophy employed in the FPGA tools. In this case, there is a layer for each of the three types of cells—standard cell, macrocell, and I/O. The standard cell layer contains rows, and the rows contain sites. Special considerations were given to the fact that modern mixed-size problems are very large (e.g., some designs possess over 2 million placeable objects, and the placement grid may have over 65 million "sites"). Since these types of designs possess uniform properties within a single row (e.g., same height, Y location, and orientation properties), many individual sites' properties could be moved into the row, thereby substantially reducing the memory required to represent a single site.

The netlist format is also similar, employing a node-edge-pin philosophy. One salient difference is that, because the legalization tool can model both global and detailed cell positions, nodes

have the ability to contain both "free-form" positions and dimensions, as well as "architectural" positions and dimensions based on the site in which they are placed. Generally, all cells begin in free-form mode, and for detailed placement, the standard cells are converted into "architectural" mode, such that all operations performed during annealing are done using `Sites` (and the dimensions of the standard cells are based on relative `Site` widths).

Note that the `Python` interpreter is not required for standard cell designs, as the architecture is built directly from the Bookshelf [4] layout files. There is also less of a need to model different types of constraints or architectures, and as such, many of the features built into the FPGA tools are not present in the standard cell tools used in this thesis.

Careful consideration was given to the handling of pin offsets, as the offsets can substantially complicate the task of bounding box caching. For run-time reasons, only the largest standard and macrocells retain their pin offsets, and the offsets on all other cells are ignored. Doing so improves the likelihood of cache hits, and can reduce run-times.

Main Data Structures used in the Standard Cell Tools

In the following code examples, only member variables are shown in order to simplify the presentation.

Architecture: Layer

The `Layer` class contains the placement grid. However, rather than being a pure "vector of vectors of `Sites`", the standard cell placement grid consists of a `Layer` which contains a `Row` which contains a `Site`. This affords some memory efficiency for the types of designs considered in this thesis.

```
class Architecture::Layer
{
protected:
        typedef std::vector< Row * > Rows_t;

        Rows_t                  _rows;                  // The rows (usually running in Y direction) in the design.
        real64                  _xmin;
        real64                  _xmax;
        real64                  _ymin;
        real64                  _ymax;
        uint32                  _maxNumSitesInRow;      // Tracks max # of sites in a row, across all rows.
        NodeType                _type;
        bool                    _isSorted;
};
```

Architecture: Row

The standard cell `Row` contains all of the common information for the `Sites` that it contains (such as their height, width, and so forth).

```
class Architecture::Row
```

```
{
protected:
        typedef std::vector< Architecture::Site >       Sites_t;

        Sites_t                 _sites;                 // The "columns" (sites) that the row contains.
        real64                  _rowHeight;             // The height of the row.
        real64                  _siteWidth;             // The width of the elements in the row.
        real64                  _siteSpacing;           // The spacing of elements in the row.
        real64                  _xorigin;               // The left-most (xmin) position of a site on this row; the row origin.
        real64                  _yloc;                  // The Y coordinate (centred) of this row.
        Architecture::Layer *   _layer;                 // The layer parent.
        sint32                  _id : 23;               // Position in the 'row' array of the layer.
        Orientation             _availableOrient : 8;   // The ways in which this node can be oriented.
        bool                    _isSorted : 1;
};
```

Architecture: Site

Unlike FPGA architectures, the standard cell `Site`s must be kept very small to reduce memory consumption, and as such, are largely bereft of unique attributes—even the X positions of the sites are computed based on their indices in their parent `Row`'s vector.

```
class Architecture::Site
{
protected:
        Node                    *_containedCell;        // The cell contained in this site.  (Sites can contain only one cell.)
        Row                     *_row;                  // Pointer to the site parent.
        sint32                  _id : 31;               // Used for determining X location of site.
        uint32                  _capacity : 1;          // The total capacity of the site.  Either 0 or 1.
};
```

Netlist: Node

The standard cell node can be either "free-form" or "architectural", meaning that a `Node` can correspond to a macrocell (whose dimensions and locations, for the purposes of academic placement tools, are relatively flexible), or to a standard cell (which must satisfy specific architectural `Site` limitations).

```
class Node
{
public:
        enum PlacedType
        {
                ToolPlaced      = 0,
                UserPlaced      = 1
        };

        enum PropertyType
        {
                PropertyType_FreeForm   = 0,
                PropertyType_Arch       = 1
        };

        struct NodePropertiesArchitecture
        // ****************************
        // Used for properties which are tied to sites in the architecture;
        // useful for cells which are being annealed.
        {
                Architecture::Site *    _site;
                sint32                  _numSiteWidths;
        };

        struct NodePropertiesFreeForm
        // *************************
        // Used for "free-form" (analytic placement-style) cell information.
        {
                real64          _height;
                real64          _width;
                real64          _xloc;
                real64          _yloc;
```

```
            Orientation     _availableOrient: 8;    // The ways in which this node can be oriented.
        };

        typedef std::vector<Pin *> Pins_t;

private:
        Pins_t                  _pins;                  // All connected pins.

        // We support either a "free-form" set of node properties or the node
        // can be tied to its architectural site.
        union
        {
                NodePropertiesFreeForm          _ff;
                NodePropertiesArchitecture      _arch;
        } _nodeProps;

        std::string             _name;                  // Name of node.
        sint32                  _id;
        Orientation             _currentOrient  : 8;    // How this node is currently oriented.
        NodeType                _type           : 4;    // Type of node.
        bool                    _isFixed        : 1;
        Node::PlacedType        _placed         : 1;    // How this particular cell has been placed.
        Node::PropertyType      _nodePropertyType : 1;  // Free-form or architecture-derived node properties.
        bool                    _isIOinCore     : 1;    // IOs in the core may be converted to macrocells; this is true in those rare
cases.
};
```

Netlist: Edge

The `Edge` class used in standard cells is effectively identical to its FPGA counterpart.

Netlist: Pin

Since architectural exploration was not a primary consideration for the standard cell tool, much of the functionality related to defined cell and pin types were removed in the interest of conserving memory. Consequently, the `Pin` class used in the standard cell annealer is a very basic `Node-Edge` connector.

```
class Pin
{
private:
        Edge *          _edge;          // Edge associated with this pin.
        Node *          _node;          // Node associated with this pin.
        real32          _offsetX;       // Offset from center.
        real32          _offsetY;       // Offset from center.
};
```

Bibliography

[1] The Python programming language. `http://www.python.org`, 2009. Current May 2009.

[2] Simplified wrapper and interface generator. `http://www.swig.org`, 2009. Current May 2009.

[3] BETZ, V., AND ROSE, J. VPR: A new packing, placement and routing tool for FPGA research. In *Field-Programmable Logic and Applications* (1997), W. Luk, P. Y. Cheung, and M. Glesner, Eds., Springer-Verlag, Berlin, pp. 213–222.

[4] CALDWELL, A. E., KAHNG, A. B., AND MARKOV, I. L. Toward CAD-IP reuse: The macro GSRC bookshelf of fundamental CAD algorithms. *IEEE Design and Test of Computers 19*, 3 (2002), 70–79.

[5] CHEN, D. T., VORWERK, K., AND KENNINGS, A. Improving timing-driven FPGA packing with physical information. In *Proceedings of FPL* (2007), pp. 117–123. Nominated for Best Paper Award.

[6] VORWERK, K., KENNINGS, A., AND GREENE, J. W. Improving simulated annealing-based FPGA placement with directed moves. *IEEE Transactions on Computer-Aided Design 28*, 2 (2009), 179–192.

[7] YANG, S. Logic synthesis and optimization benchmarks, version 3.0. Tech. rep., Microelectronics Center of North Carolina, 1991.