On The Engineering of a Stable Force-Directed Placer

by

Kristofer Vorwerk

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2004

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

Analytic and force-directed placement methods that simultaneously minimize wire length and spread cells are receiving renewed attention from both academia and industry. However, these methods are by no means trivial to implement—to date, published works have failed to provide sufficient engineering details to replicate results.

This dissertation addresses the implementation of a generic force-directed placer entitled FDP. Specifically, this thesis provides (1) a description of efficient force computation for spreading cells, (2) an illustration of numerical instability in this method and a means to avoid the instability, (3) metrics for measuring cell distribution throughout the placement area, and (4) a complementary technique that aids in minimizing wire length. FDP is compared to Kraftwerk and other leading academic tools including Capo, Dragon, and mPG for both standard cell and mixed-size circuits. Wire lengths produced by FDP are found to be, on average, up to 9% and 3% better than Kraftwerk and Capo, respectively. All told, this thesis confirms the validity and applicability of the approach, and provides clarifying details of the intricacies surrounding the implementation of a force-directed global placer.

Acknowledgments

My heartfelt thanks extend to Dr. Andrew Kennings—without his help, guidance, and encouragement, this work simply would not be. My thanks also extend to Dr. Tony Vannelli, Mom, Katrina, Andréa, and John, for their continued support.

To Katrina.

Tant que la lecture est pour nous l'initiatrice dont les clefs magiques nous ouvrent au fond de nous-mêmes la porte des demeures où nous n'aurions pas su pénétrer, son rôle dans notre vie est salutaire.

Marcel Proust, *Sur la lecture*

Table of Contents

List of Tables

List of Illustrations

Chapter 1

Introduction

The development of a modern integrated circuit (IC) can be a daunting task. More and more, designers must balance the constraints of power and performance, all while remaining within the confines of short design cycles. It is for this reason that design automation tools have come to play an increasingly important role in the development of ICs.

As circuits have grown in size, performance has become limited by the delay of the interconnect rather than the switching speed of logic elements. Consequently, research in computer-aided design (CAD) algorithms has focused much attention on the minimization of wire length and critical path delay. Yet, these algorithms can be complicated—the optimization problems that must be solved are often NP-complete [1–3]. Thus, the optimal solutions to many of these problems cannot be found in polynomial time; rather, heuristic methods are employed to approximate the optimal solutions. This thesis presents a detailed discussion of one such heuristic for computer-aided circuit placement.

1.1 VLSI CAD Flow

The VLSI 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 follows

a series of steps, as shown in Figure 1.1.

Synthesis is an automatic method for converting the high-level design to a gate-level netlist consisting of interconnected gate-level macro cells [4, 5]. In deriving this netlist representation, an attempt is made to minimize the number of gate-level cells using logic optimization. The circuit is then mapped into cells based on the available technology library.

The inputs to the *placement* phase are the module description, consisting of the shapes, sizes, and terminal locations, and the netlist, describing the interconnections between modules. The output is a list of x- and y-coordinates for all modules [6]. Placement seeks to position cells in valid locations (without overlap) while optimizing chip area, wire length, and critical path delay. Area is typically minimized to be able to fit more logic functionality into a given chip, while wire length is minimized to reduce the interconnect delay. In some cases, secondary performance measures, such as the preferential minimization of delay in a few critical nets, may also be employed, at the cost of an increase in total wire length [6].

Since module placement is NP-complete [1–3], it is not possible to find an optimal placement of cells in polynomial time. Trying to find such an exact solution would take time proportional to the factorial of the number of modules [3]; consequently, a heuristic approach must be used to efficiently search through the candidate placements. The quality of this heuristic can largely determine the performance and area requirements of the final, integrated circuit.

Placement usually consists of both *global* and *detailed* phases—the former strives to locate cells within the general vicinity of where they should be fixed, and the latter ensures that final cell positions are *legal*. Iterative techniques are often applied during or after detailed placement to further improve results.

After the cells have been placed, the circuit is then *routed*. This process establishes the pin-to-pin connections between cells. Finding an optimal routing given a placement is also a NP-complete problem [6], although a number of heuristic approaches exist (c.f. [7, 8]) which can find very good, admissible routings in polynomial time.

Placement heuristics often work in consort with routing by considering net congestion when

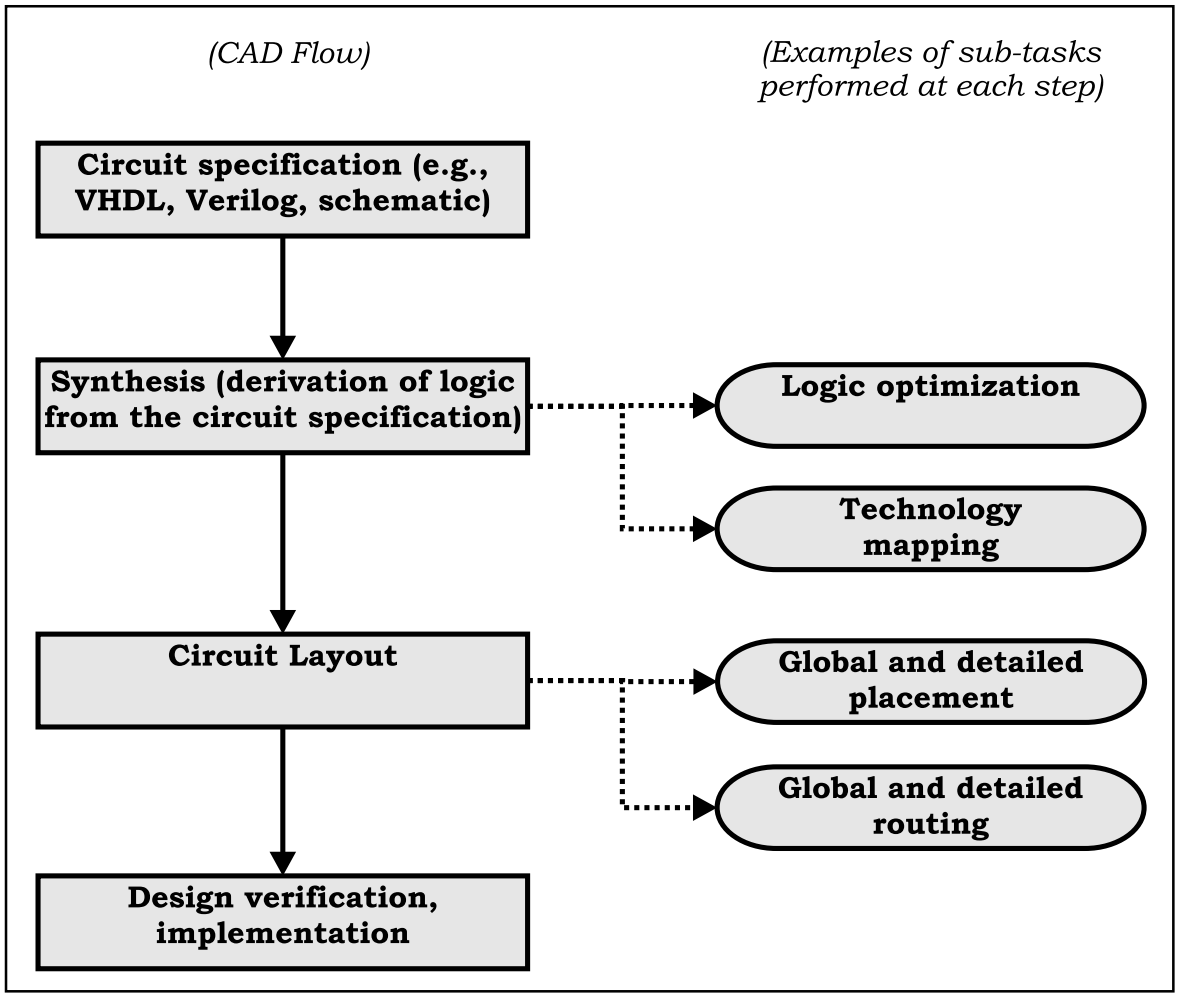


Figure 1.1: Traditional view of the VLSI CAD flow.

placing cells. Figure 1.2 illustrates the nets of a small integrated circuit after the application of a placement heuristic, compared to the those net lengths resulting from a random assignment of cells. In this example, fixed pads are located along the periphery of the chip, while movable cells are located inside the chip. The nets in the random placement are highly congested, while the opposite is true of the circuit which has undergone module placement. The shorter, less congested nets of the latter are more likely to route (and use fewer layers of metal), and are also more likely to offer a faster overall implementation. An example of the routed version of the circuit is provided in Figure 1.3.

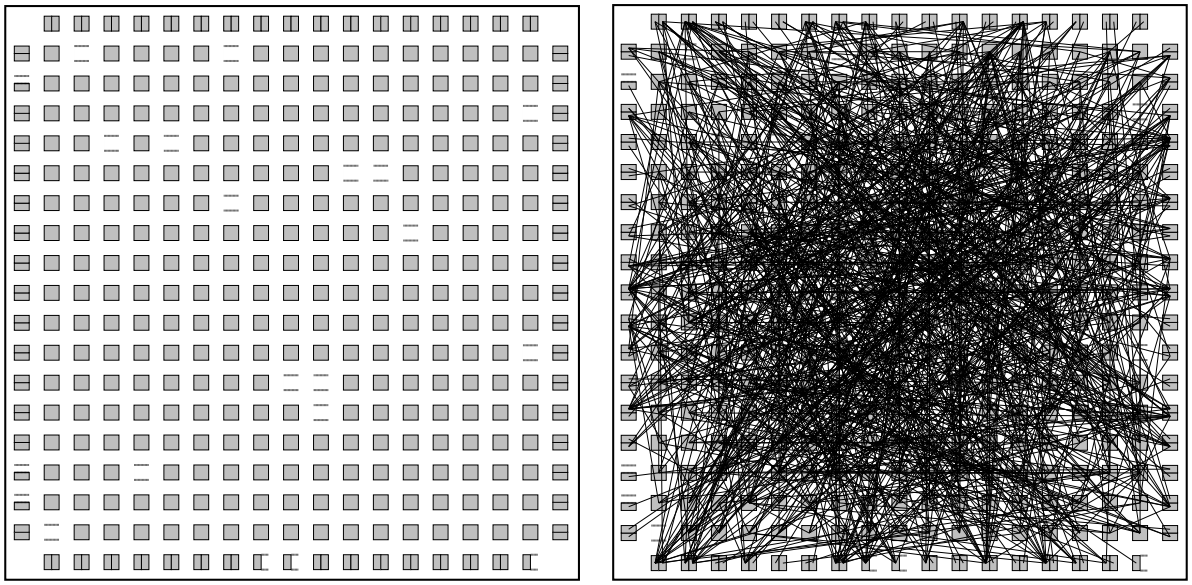
The final steps in the VLSI CAD flow are to verify the circuit, ensuring that the layout meets system specifications and fabrication (or programming) requirements. This step often consists of a "design rule check" and "circuit extraction". In these procedures, the implementation is checked to ensure that it meets fabrication constraints and that the functionality of the final implementation matches the original specification. Once verified, the design may be implemented based on the desired *design style*, described in Section 1.2.

1.2 Design Styles for Modern VLSI CAD

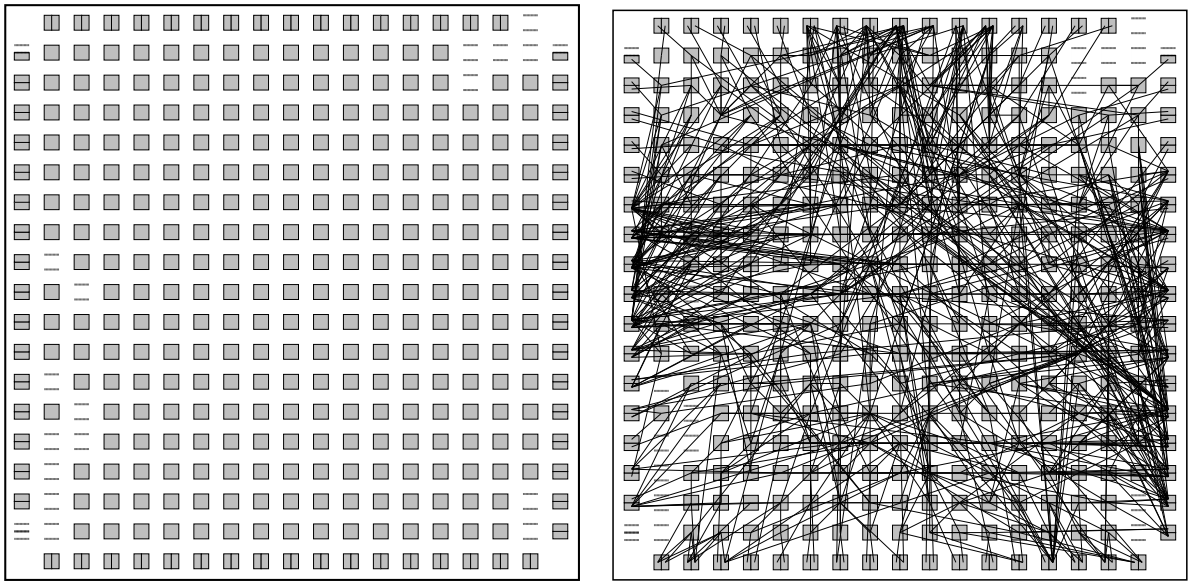
Computed-aided design tools make it possible to automate the entire layout process for VLSI designs. This has been made possible through the use of restricted models and *design styles* which reduce the complexity of the circuit layout. Two design styles which are typically used in modern CAD flows include *standard cells* and *field programmable gate arrays* (FPGAs).

1.2.1 Standard Cell Circuits

A standard cell is a logic module with a pre-designed internal layout. These cells have a fixed height but different widths, depending on the functionality of the module [6]. 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



a. Random assignment of cells and I/O pads, and the resultant nets.



b. Locations of cells and the resultant nets after placement.

Figure 1.2: Illustration of cells, I/O pads, and nets due to both a random assignment and a placement heuristic, prior to routing. Note how the placement heuristic rearranged cells to visibly minimize wire length and net congestion. The circuit in this example is very small by modern standards and is provided only for illustrative purposes. In this case, there are 274 logic cells, 65 I/O pads, and 339 nets. This problem instance is known as `e64` and was captured using the `VPR` FPGA placement tool, described in Chapter 2.

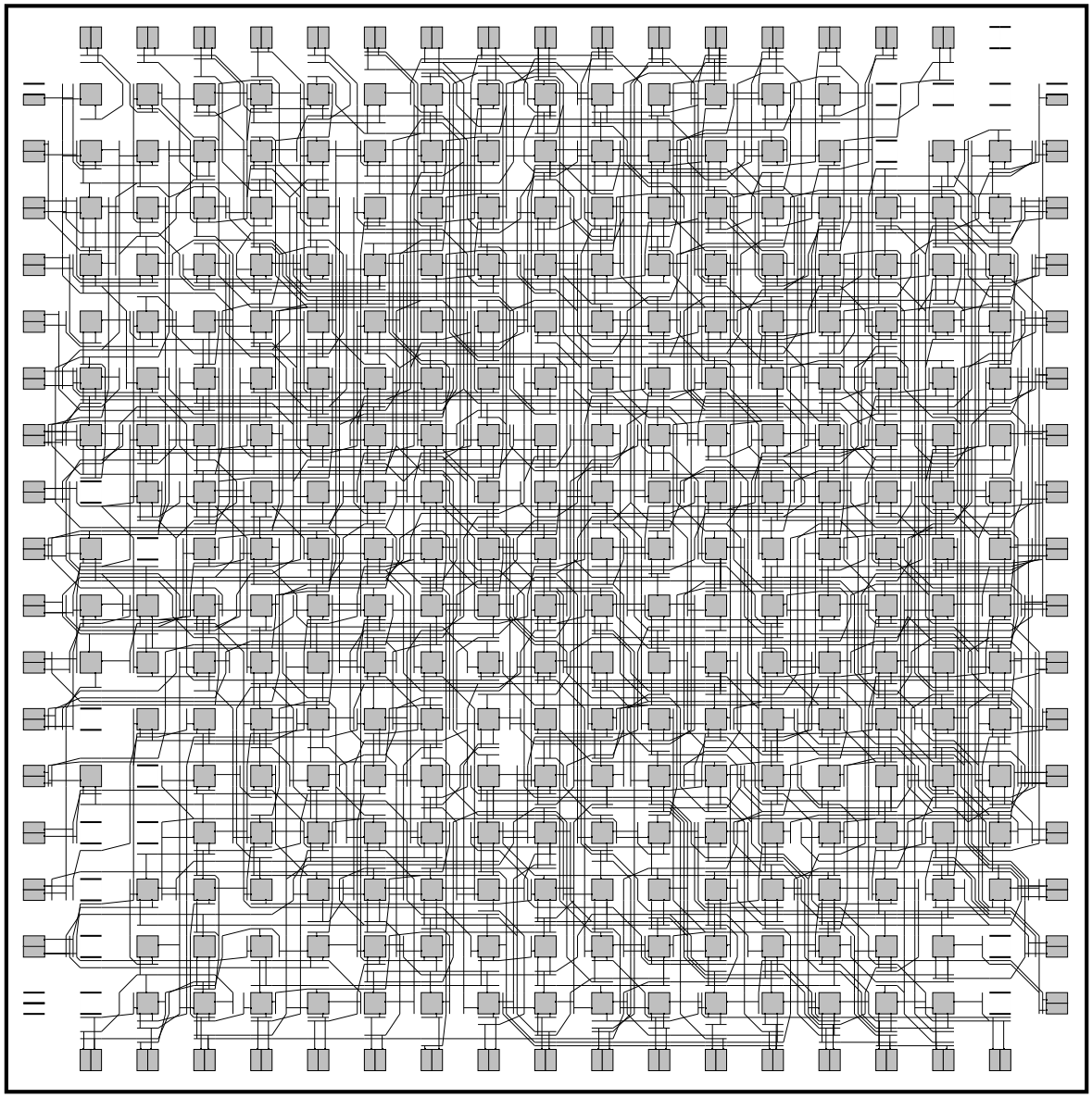


Figure 1.3: Illustration of the resources used to route the design of Figure 1.2 (b).

more layers of metal available for routing, channels are not typically required. Logic modules connect to fixed pads (terminals) along the edges of the chip. *Macro cells* 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 macro cells are referred to as *mixed-size* designs. Figure 1.4 shows an example of a 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 1.5. There are three main types of configurable elements: logic blocks, input/output (I/O) blocks, and interconnects.

Logic blocks permit the implementation of a designer's logic. Logic blocks may possess look-up tables and flip-flops which allow a designer to implement combinatorial and synchronous logic. In addition to simple logic blocks, modern FPGAs may also implement programmable RAMs, carry-chains, embedded processors, or other macro-sized blocks.

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 [7, 9]. FPGAs are customized by loading configuration data into internal static memory cells. Stored values in these cells determine the logic functions and interconnections in the FPGA.

An example of an FPGA's routing resources, as well as a sample programming of the resources is shown in Figure 1.6. The interconnect, I/O cells, and logic cells are visible in this figure, as well as the resultant, programmed interconnect. 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 lookup table and a flip-flop. In the figure, pins 0 to 3 of each logic cell act as inputs, pin 4 represents an output, and pin 5 represents the global clock input.[1]

1 The routing resources for the global clock are not shown. Since clocks are generally distributed to every logic cell

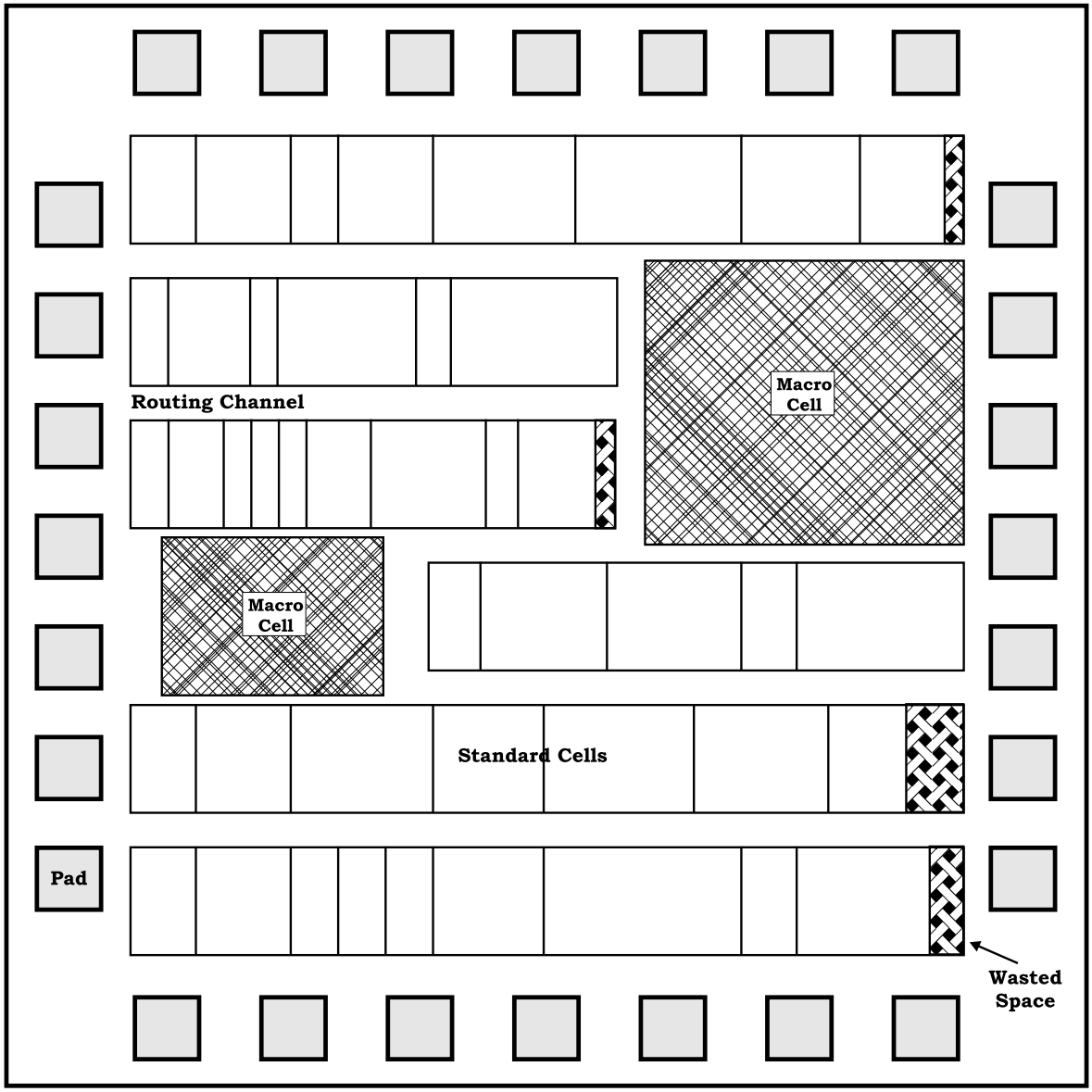


Figure 1.4: Diagram of a mixed-size standard cell layout. Standard cells are placed in rows, but macro cells are not subjected to this constraint. "Whitespace" (unused, wasted space) may also be present in the design. Fixed pads, representing connections to the I/O terminals, are typically located along the periphery of the chip.

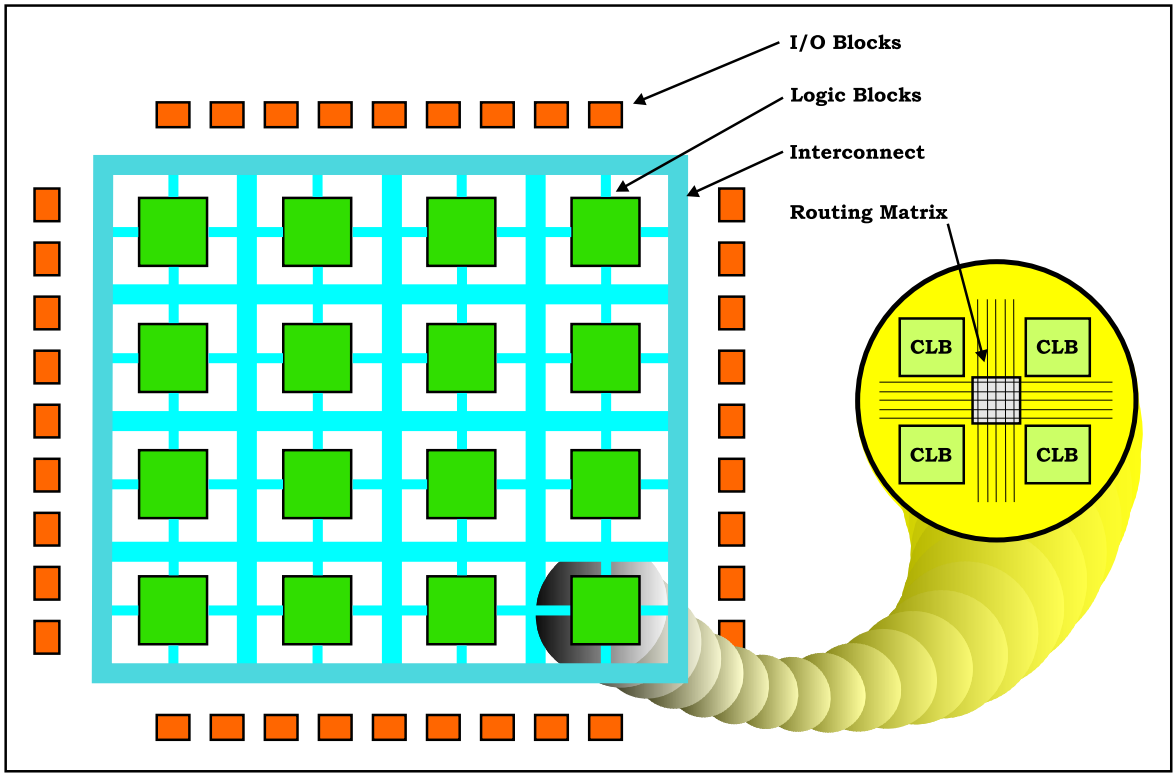
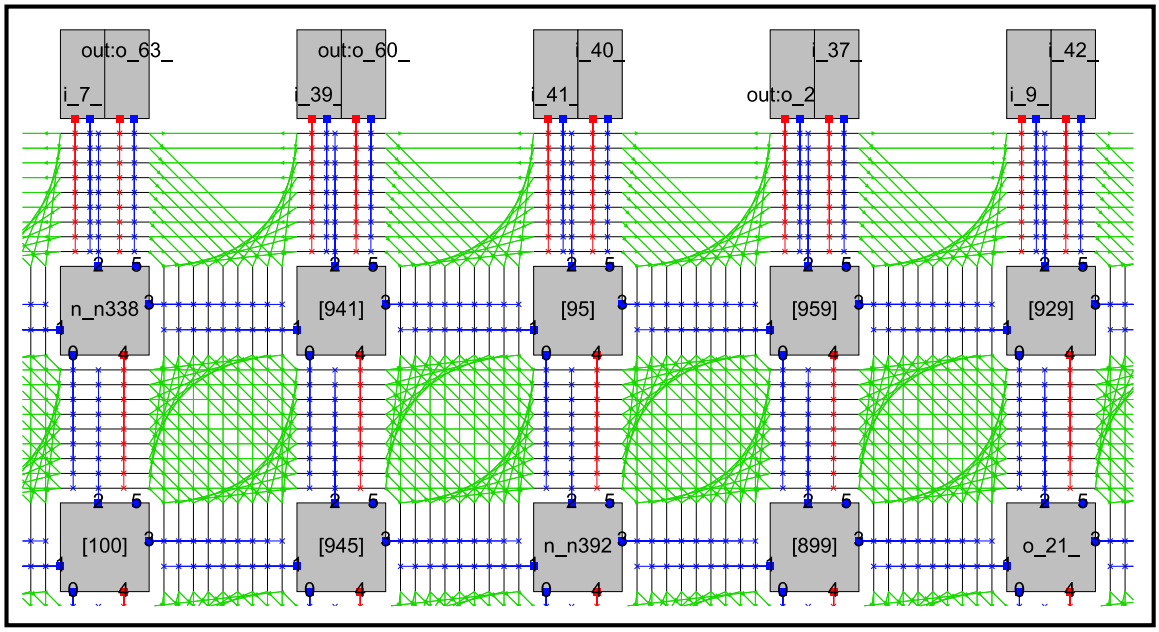
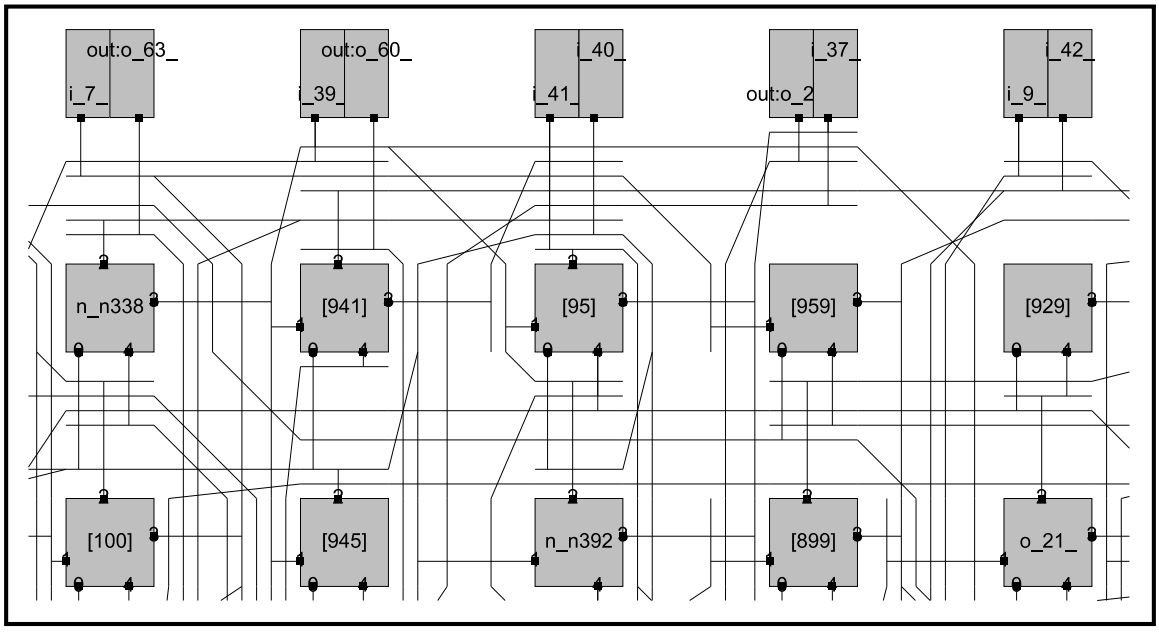


Figure 1.5: Diagram of a simplified FPGA architecture. Internal connections are made by programming the switching points to connect metal segments.



a. Magnified view of the routing resources in an FPGA.



b. Programmed interconnect.

Figure 1.6: Magnified view of a programmable interconnect in an FPGA. The top row of each diagram contains I/O blocks, while the second and third rows contain movable cells.

1.2.3 Placement

With respect to placement, standard cell and FPGA designs differ primarily in their need for alternative legalization strategies. In both cases, similar global placement heuristics may be used to locate cells, while a different detailed placement strategy would be used to ensure that overlap is fully removed subject to the constraints and characteristics of each architecture. Standard cell circuits, for example, typically require that all overlap be removed. FPGA circuits, on the other hand, require that cells be placed in valid (pre-assigned) locations, and, furthermore, that all constraints on cell type and wiring resources be satisfied. In both cases, the same global placement heuristic can be used to achieve a fairly non-overlapping placement, while a different legalization strategy can be employed to finalize the layout.

1.3 Conventions and Terminology

In this thesis, the terms *module*, *cell*, and *node* are used to describe a standard or macro cell. *Macro cell* 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 [6]. The term *placer* refers to a tool which implements a heuristic to place cells.

1.4 Statement of Thesis

The need for better CAD tools, and especially placement heuristics, has become increasingly important as circuits and placement instances have grown in size. This dissertation describes a practical implementation of a global, force-directed placer capable of accommodating modern VLSI problems. While the literature provides a good starting point for the development of such a heuristic, numerous additional techniques are required to stabilize the method and to improve

in an FPGA, they are routed along dedicated, high-speed lines.

the quality of results. None of the recent literature on force-directed placement (c.f. [10–13]) has addressed the stability and quality issues broached here. Specifically, this dissertation presents and expounds upon:

- a means of efficiently computing forces for cell spreading;
- techniques for assessing cell distribution throughout the placement area;
- a method of preventing the destabilization of a placement due to numerical instability; and,
- a method for direct minimization of wire length that significantly improves the overall quality of the placement.

The rest of this work is organized as follows. Chapter 2 presents an overview of modern placement heuristics. Chapter 3 introduces the concept of force-directed placement and discusses the engineering details of the framework; moreover, it describes the additional techniques that were developed for enhancing placement quality. Chapter 4 touches upon this dissertation's strategy for detailed placement and presents results comparing the force-directed placer to other modern approaches. Finally, Chapter 5 offers concluding remarks and discusses future directions.

Chapter 2

Background

The quality of the global placement heuristic used in the CAD flow 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 [14]. On the other hand, the demand for higher-quality placement techniques must also be balanced with the need for shorter running times—some heuristics, which worked fine twenty years ago, are no longer able to keep pace with the size of modern designs [6, 14, 15]. In this chapter, the three most common types of heuristics for global placement—search-based, partitioning-based, and analytic methods—are examined, and recent advances in each class are discussed.

2.1 Overview

A circuit is typically modeled as a hypergraph $G_h(V_h, E_h)$ with vertices $V_h = \{v_1, v_2, \ldots, v_n\}$ representing cells and hyperedges $E_h = \{e_1, e_2, \ldots, e_m\}$ corresponding to signal nets. Vertices are weighted by cell area while hyperedges are weighted according to criticalities or multiplicities. Vertices are either free or fixed. Cell placements in the x- and y-directions are captured by placement vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$.

A placer seeks to position cells from the hypergraph in valid, non-overlapping locations within a *placement region* while minimizing chip area and critical path delay. Ideally, a placer also makes

an effort to ensure that all wires can be routed. Additional objectives may also attempt to minimize power dissipation or reduce cross talk between signals [16].

Placement objectives are often difficult to formulate and computationally difficult to satisfy optimally in present day integrated circuits. As a result, placement tools use approximations to these objectives. Some commonly-used approximations attempt to:

- minimize the total estimated interconnect length;
- meet the timing requirements for critical nets; and,
- minimize the interconnect congestion [16].

Wire length is one of the most commonly-employed measures of quality in VLSI CAD because, by minimizing wire length, the delays of the interconnect are also reduced.

During placement, the minimum wire length of a net can be measured by finding the minimal Steiner tree connecting the pins of the net. However, given that finding the minimum Steiner tree is NP complete [3], and that the number of nets in modern circuits can be very large[1], this method of measuring wire length is impractical. Finding a minimum spanning tree can also be too computationally demanding, owing, again, to the large number of nets in modern circuits. Instead, an approximate, easy-to-calculate measure of wire length, closely correlated with the final interconnect length, is required [16]. The measurement most commonly used in modern tools is the *half perimeter wire length* (or HPWL) of the minimum rectangle enclosing all cells on a net.

Generally, search-based placement methods are able to minimize HPWL directly, while partitioning- and analytic-based methods minimize approximations to HPWL, such as the minimum net cut or quadratic (instead of Euclidean) wire length. As a result of using an approximation in their objective formulation, analytic and partitioning-based placers can process very large problem instances. It is for this reason that hybrid methods which combine more than one placement approach often use an approximative technique to encourage quick cell placement, while interleaving calls to a direct HPWL-minimizing algorithm to improve the quality of results.

1 Some placement benchmarks, such as `ibm18` (see Chapter 4), possess more than two hundred thousand nets.

2.2 Search-Based Placement

Search-based placement methods involve iterative improvement of an existing solution. For example, simulated annealing-based placers, such as `TimberWolf` [17, 18] and `VPR` [7], produce placements using stochastic search (c.f. [19]). 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" [6]. Genetic algorithms are no longer used in modern CAD flows, and are not treated here.[2]

2.2.1 Simulated Annealing

Simulated annealing is perhaps the most well-developed, well-studied method for module placement. It can be very 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. Due to the ever-evolving nature of the FPGA marketplace, vendors must continually adapt their placement heuristics to account for new product generations. Moreover, 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 algo-

2 The reader is referred to [6] for more information about genetic placement.

rithm. In this approach, the heuristic periodically accepts moves that result in an increase in the cost—this ensures that the method will not get 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 [6]. 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 [6]. 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 a sufficiently long time with a proper cooling schedule, simulated annealing is guaranteed to converge to the global minimum [20].

Simulated annealing derives its name from the annealing process in metals [19]. 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 [6]. 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 general 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.[3] A range-limiting function may be applied to ensure that cells are not moved further than a specified distance from the target location [6]. 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, then the new placement may still

3 Cells can even be rotated or mirrored as part of the perturbation.

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 $T_{i+1} = 0.1T_i$.

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" [6]. For example, in `TimberWolf` [21], cells are chosen randomly and either interchanged or displaced to a random location on the chip. The algorithm performs best when the number of displacements is between three and eight times the number of interchanges [21].

The cost function employed by `TimberWolf` accounts for wire length and 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.

2.3 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

```
1  Procedure: SIMULATED ANNEALING
2  Input: A netlist, N
3  begin
4    Initialize variables;
5    Generate a random placement of the cells in N;
6    while outer_loop_count < MAX_OUTER_PASSES and T > MIN_TEMP do
7      inner_loop_count ← 0;
8      while inner_loop_count < MAX_INNER_PASSES do
9        Perform a random perturbation of the placement;
10       ΔC ← the change in cost of the placement;
11       if ΔC < 0 or the probability function accepts the move then
12         Accept the new placement;
13       else
14         Reject the new placement (and restore the previous cell location);
15       fi
16       T ← decreased value based on cooling schedule;
17       inner_loop_count ← inner_loop_count + 1;
18     od
19     outer_loop_count ← outer_loop_count + 1;
20   od
21 end
```

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

subregions, reformulating constraints, and cutting the netlist—such that good solutions to smaller instances (subproblems) combine into good solutions of the original problem" [22]. 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.

Modern partitioning-based placers decompose the netlist using minimum-cut hypergraph bi-partitioning. Quadri-section techniques [23] 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 [6]. 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" [6]. 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 [6].

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 [22]; otherwise, relaxed balance constraints can lead to uneven area utilization and overlapping placements [22, 24].

Partitioning is typically performed using an iterative, multi-level Fiduccia-Mattheyses (FM) heuristic [25, 26]. The Kernighan-Lin (KL) heuristic [27, 28] is also used for hypergraph bi-partitioning. For example, the popular multi-level partitioner, `hMetis` [29–31], employs FM for large partitioning instances, while KL is used when the instances are smaller than a threshold parameter.

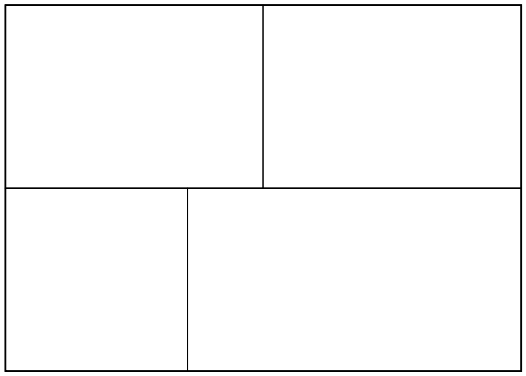
Once a partitioned block is sufficiently small (or contains too few cells), partitioning-based placers use an alternate, *end-case* algorithm to finalize cell locations. Tight balance constraints and a potentially large variation in standard cell sizes makes small partitioning instances difficult for 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" [22]. Consequently, a branch-and-bound strategy is typically employed for small partitioning instances (c.f. [22, 32]).

2.3.1 Recent Advances

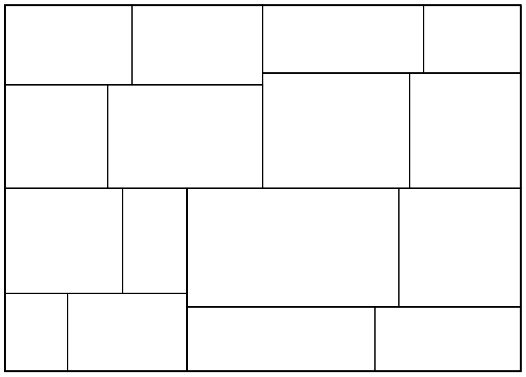
With the advent of multi-level hypergraph partitioning in 1997 [30, 31], the quality of cuts generated by partitioners improved significantly, and by extension, so did the quality of VLSI placements.[4] Since then, hundreds of papers have undertaken the task of improving upon partitioning-based techniques; the following is a selection of some of the most pertinent works.

In [22], 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

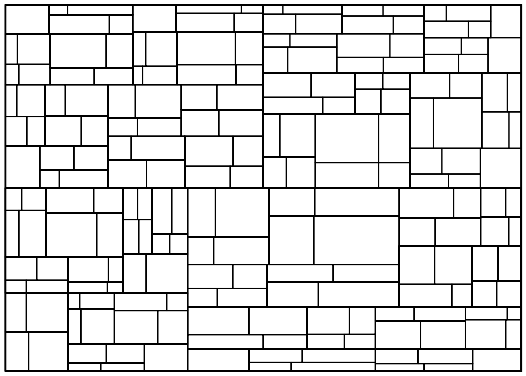
4 An excellent review of partitioning and its applications to placement prior to 1995 is given in [24].



a. One level.



b. Two levels.



c. Up to three 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.

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

Figure 2.3: High-level pseudocode for a top-down partitioning based placement technique [22].

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 [33], 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 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 [33]. This technique forms the basis for the `BonnPlace` [34] placement tool.

Caldwell et. al. introduce a recursive bisection placement tool in [35]. This paper builds upon the authors' previous work on multi-level hypergraph partitioning and end-case placement (c.f. [22, 25, 32]), and describes the implementation of the first commercial-quality, academic partitioning-based placer, `Capo`. Since then, `Feng Shui` [36] has emerged as another popular 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 to satisfy overlap constraints.

In [37], the authors describe a means of augmenting partitioning-based placement using analytic strategies. (Analytic methods are discussed more thoroughly in Section 2.4.) 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 [38]. 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 (c.f. [39]). 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.

In [40], modifications to the `Feng Shui` placer to handle mixed-size designs are described. The traditional recursive bisection approach employed in `Capo` typically "shreds" macro cells into standard cell sizes and connects the shredded components via high edge weights. The average location of the shredded cells is taken as the final positions of the macro cells. However, in [40], macro cells are processed simultaneously by simply not performing shredding and allowing cut lines to be placed in any location. A minimum-movement legalization strategy is then employed to remove overlap.

Kahng and Reda, in [41], 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 [38]. 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.

2.4 Analytic Placement

Analytic placement methods (c.f. [13, 34, 39, 42–44]) use linear or quadratic optimization to place cells. While linear programming formulations themselves are generally not employed for global placement, various other techniques have been used to approximate a linearized objective (c.f. [45–49]).

In analytic placers, circuit hypergraphs are typically transformed into graphs in which each hyperedge is represented by a set of equally-weighted edges. The *star model* adds a new centre vertex and represents the original net by edges connecting the centre to the previously existing vertices. The *clique model* connects all pairs of vertices incident to the original hyperedge by equally-weighted edges. A clique and star model for a five-pin net are shown in Figure 2.4. Clique models of large hyperedges become prohibitively expensive due to the quadratic edge count. Consequently, large edges are either dropped completely, or a combination of clique and star models are employed in which cliques are used to model small hyperedges and stars are used to model large hyperedges.[5]

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.

In [50], 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 [51].

5 The notion of what may be a "large" or "small" hyperedge can vary between analytic methods. Experimental results for net sizes in this dissertation are provided in Section 3.2.2.

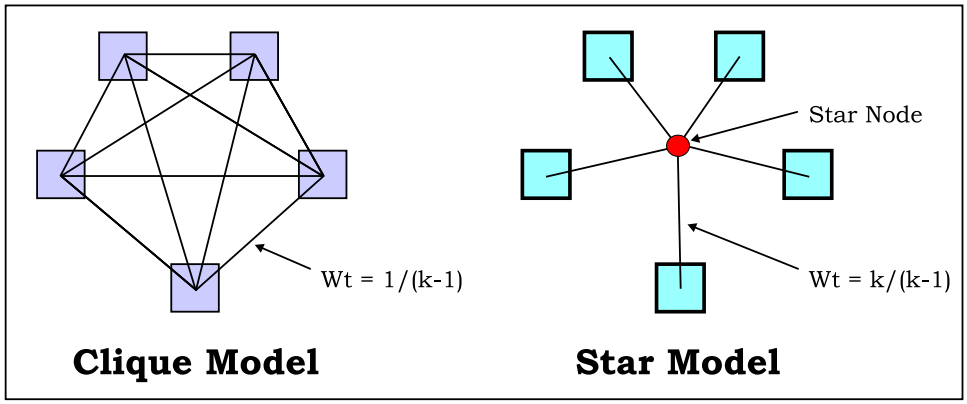


Figure 2.4: A comparison of the clique (left) and star (right) net models for a five-pin net.

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

$$\phi = \sum_{i,j} \mathbf{C}_{ij}\mathbf{D}_{p(i)p(j)} \tag{2.1}$$

is minimized [6].

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 [6]. Thus, ϕ gives the total wire length for the circuit, which is to be minimized [6]. 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 [16, 52]. 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 partitioning or "spreading forces", as discussed in Section 2.5 and Section 3.1.[6]

2.4.1 Eigenvector Placement

Hall [50] formulated the quadratic placement problem as follows. Let **C** be the connection matrix. Let c_i be the sum of all elements in the i^{th} row of **C**. A diagonal matrix **D** is defined such that

$$d_{ij} = \begin{cases} 0, & \text{if } i \neq j, \\ c_i, & \text{if } i = j. \end{cases}$$

The matrix **B** is defined as

$$\mathbf{B} = \mathbf{D} - \mathbf{C}.$$

Recall, from Section 2.1, that vectors **x** and **y** are typically used to capture the resultant placement. Hall proved that Equation (2.1) can be rewritten as

$$\phi = \mathbf{x}^T\mathbf{B}\mathbf{x} + \mathbf{y}^T\mathbf{B}\mathbf{y}$$

subject to the constraints that $\mathbf{x}^T\mathbf{x} = 1$ and $\mathbf{y}^T\mathbf{y} = 1$. Hall then showed that the eigenvectors of the matrix **B** are the solutions to the placement problem—that is, the eigenvectors of **B** give the *x*- and *y*-coordinates of all modules. In a two-dimensional placement problem, the *x*- and *y*-coordinates are given by the eigenvectors corresponding to the two smallest, nonzero eigenvalues.[7]

---

6 The majority of the discussion of force-based placement is deferred until Chapter 3.

7 Using the two smallest, nonzero eigenvalues avoids the trivial solution of $\mathbf{x}_i = 0$ (for all i).

2.4.2 Quadratic Placement

An alternative quadratic formulation was introduced in [39]. In this approach, the overall method for minimizing wire length is accomplished by solving the quadratic optimization problem (x-direction only) given by

$$\min_x \left(\sum_{i,j} a_{ij}(x_i - x_j)^2 \right) = \min_x \frac{1}{2}\mathbf{x}^T \mathbf{Q}_x \mathbf{x} + \mathbf{c}_x^T \mathbf{x} + \mathbf{d}_x \tag{2.2}$$

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,

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

In this formulation, cell overlap is ignored, and the vector $\mathbf{x}$ provides only relative cell locations. (The resultant placement would be infeasible and would have to be legalized.) Pseudocode for setting up the Hessian and cost vector (i.e., the objective function) is provided in Figure 2.5 and Figure 2.6. An example of the highly-overlapping nature of a quadratic placement is shown in Figure 2.7.[8]

Studies have demonstrated that quadratic optimization, although most efficient, tends to produce placements of inferior qualities. Better results may be achieved using iterative quadratic

8 The reader is referred to [39, 42, 50] for more information about the quadratic problem formulation.

```
1   Procedure: OUTERLOOP
2   Input: A netlist, N, Hessian Q, cost vector c
3   begin
4     Q ← 0;
5     c ← 0;
6     for each edge e ∈ N do
7       k ← number of cells on edge e;
8
9       // Edge weights are typically calculated as w = 1/(k−1).
10      w ← weight for edge e;
11
12      if k ≥ 2 and k < MAX_CLIQUE_SIZE then
13        // Build using clique net model.
14        for each cell p_i on e do
15          for each cell p_j | j > i on e do
16            call INNERSETUP( Q, c, p_i, p_j, w );
17          od
18        od
19      else if k ≥ MAX_CLIQUE_SIZE then
20        // Build using star net model. Weights are multiplied by k.
21        w ← k × w;
22        p_i ← star node for edge e;
23        for each cell p_j | j ≠ i on e do
24          call INNERSETUP( Q, c, p_i, p_j, w );
25        od
26      fi
27    od
28  end
```

Figure 2.5: Simplified pseudocode for the outer loop used in setting up the quadratic placement objective for a hybrid clique/star model.

```
1   Procedure: INNERSETUP
2   Input: Hessian Q, cost vector c, cells p_i, p_j, edge weight w
3   begin
4     // Note that the indices which map into Q also map similarly into c.
5     m_i ← index that maps p_i into matrix Q;
6     m_j ← index that maps p_j into matrix Q;
7
8     if p_i is movable and p_j is movable
9       Q(m_i, m_i) ← Q(m_i, m_i) + w;
10      Q(m_j, m_j) ← Q(m_j, m_j) + w;
11      Q(m_i, m_j) ← Q(m_i, m_j) − w;
12      Q(m_j, m_i) ← Q(m_j, m_i) − w;
13    else if p_i is movable and p_j is fixed
14      Q(m_i, m_i) ← Q(m_i, m_i) + w;
15      c(m_i) = c(m_i) + w × location of cell p_j;
16    else if p_i is fixed and p_j is movable
17      Q(m_j, m_j) ← Q(m_j, m_j) + w;
18      c(m_j) = c(m_j) + w × location of cell p_i;
19    else
20      // Both cells are fixed, so do nothing.
21    fi
22  end
```

Figure 2.6: Simplified pseudocode for setting up a quadratic placement objective based on [39]. This function shows how to set-up the Hessian matrix **Q** and cost vector **c** for one direction only.

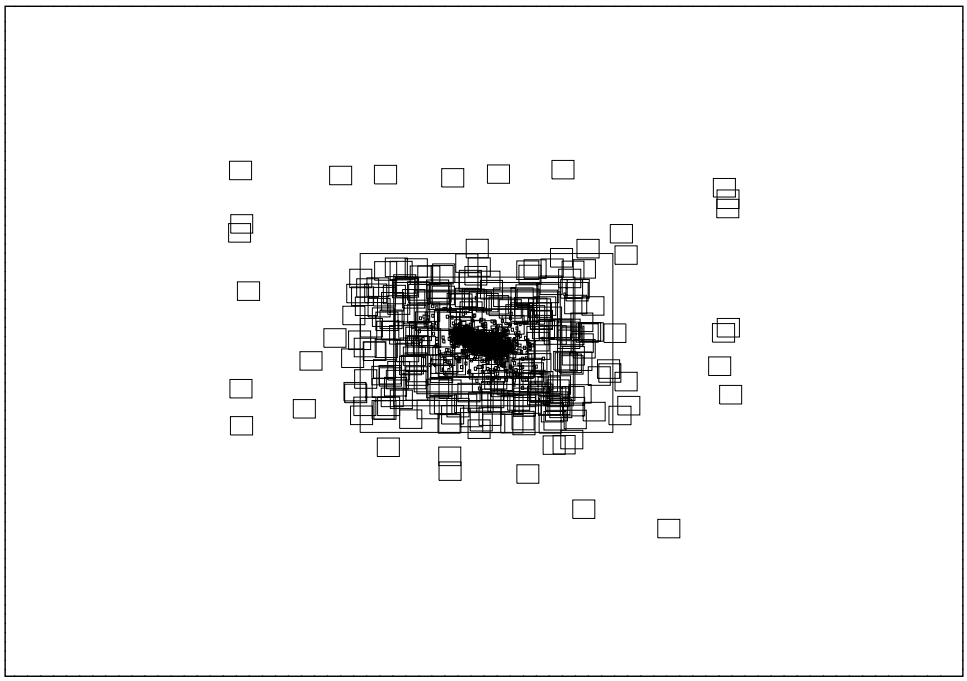


Figure 2.7: Quadratic placement for a mixed-size problem with approximately twenty-seven thousand cells. I/O pads, fixed along the periphery, "pull" some of the cells outward from the centre, but the placement is still far from legal.

optimization with re-weighting at the expense of potentially significant increases in run-time. For instance, in [44], a series of quadratic problems with re-weighting given by

$$\min_{x^v} \sum_{ij} \frac{a_{ij}}{|\mathbf{x}_i^{v-1} - \mathbf{x}_j^{v-1}|} (x_i^v - x_j^v)^2$$

are solved to yield cell locations. Here, $\mathbf{x}^{v-1}$ and $\mathbf{x}^v$ denote the vectors of vertex locations at iterations $v-1$ and v, respectively. Thus, a quadratic objective function is used for efficient optimization, but re-weighting is used to approximate linear rather than quadratic distances.

Analytic placement is ideally performed using unconstrained, strictly convex formulations that are amenable to efficient solutions using Newton-type solvers. It is these efficient and "easily solvable" formulations that enable analytic methods to be used on modern, large-scale placement problems.

2.4.3 Recent Advances

In [47], Kennings and Markov establish that net models lead to a loss in solution quality. To address this loss, the authors introduce an analytic heuristic known as BoxPlace that does not require net models and permits a direct inclusion of non-linear delay terms. BoxPlace finds the median location of nets and moves cells directly to a point of minimal HPWL.

In [53], Kahng et. al. introduce a continuous optimization problem to improve timing within the context of a top-down placement algorithm. The reduction of net delay is accomplished by the minimization of maximal slack; that is,

$$\phi = \min \left(\max_{\pi} \sum_{e \in \pi} \frac{1}{c_\pi} \times \text{delay}(e) \right)$$

for all paths π and per-path weighting, c. The minimization of this objective, which can be accomplished by linear or non-linear programming (depending on net delay models) is then shown to be solvable in linear time using linear wire length delay objectives [53].

Obenaus and Szymanski introduce an alternative means of dealing with overlapping cells from a quadratic placement in [54]. In this approach, cells are initially positioned along the outer periphery of the placement region. A quadratic formulation is then used to draw the nodes closer to the centre of the placement region—effectively implementing a "gravitational" force. This gravitational tendency is counter-balanced by slowing cell movement using force-based iterations and by binning and rescaling parts of the placement region to arrive at an approximately uniform cell distribution.

In [55], the authors describe and extend a patent originally presented in [56]. In this work, the placement area is divided into grid bins, with the objective being to equalize the total cell area in every grid subject to a penalty function. The penalty function may be formulated as:

$$\rho = \sum_{\text{bin } b} \left(\text{TotCellArea}(b) - \text{AvgCellArea}(b)\right)^2.$$

Since ρ is not smooth or differentiable, it is difficult to minimize. The authors account for this difficulty by proposing a bell-shaped potential function, given by

$$\text{Potential}(c,b) = \alpha(c) \cdot f(|c_x - b_x|) \cdot f(|c_y - b_y|)$$

for each bin b and cell c, whose (x,y) centre is (b_x, b_y) and (c_x, c_y), respectively. In this formulation,

$$f(q) = \begin{cases} 1 - \frac{2q^2}{r^2}, & 0 \le q \le \frac{r}{2} \\ \frac{2(q-r)^2}{r^2}, & \frac{r}{2} \le q \le r \end{cases}$$

where r controls the radius of the cells' potentials and $\alpha(c)$ is a proportionality factor used to ensure that

$$\sum_{\text{bin } b} \text{Potential}(c,b) = \text{Area(c)} \; \forall \; c \in V.$$

This yields the following form of the optimization problem, which can be solved efficiently using

a conjugate gradient technique:

$$\text{Penalty} = \sum_{\text{bin } b} \left(\sum_{\text{cell } c} \text{Potential}(c,b) - \frac{\text{TotCellArea}}{\text{NumGridBins}} \right)^2 .$$

The authors then extend the objective function to account for wire length using the convex log-sum-exponential [57] approximation of HPWL,

$$\text{WL} = \alpha \times \left(\ln(\sum e^{\frac{x_i}{\alpha}}) + \ln(\sum e^{-\frac{x_i}{\alpha}}) \right)$$

for a net t with pin coordinates $(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)$ and smoothing parameter α [47].[9] Finally, the wire length and penalty objectives are linearly weighted to give the objective function: $\beta \cdot \text{WL} + (1-\beta) \cdot \text{Penalty}$. The objective is solved repeatedly, with the weight β controlling the "preference" for overlap removal versus cell spreading in each iteration.

`FastPlace` [15] augments quadratic placement with heuristics for spreading cells and performing local refinement. The authors describe a cell shifting technique which removes overlap from a placement by first creating an equal and an unequal bin structure over the placement area for the *x*-direction. The unequal bin structure is based on the utilization of cells in a row. Every cell belonging to a particular bin in the regular bin structure is then linearly mapped to the corresponding bin in the unequal structure. As a result, cells in bins with a high utilization shift in a way so as to reduce the bin utilization (and the overlap between cells) [15]. A similar strategy is then applied to columns in the *y*-direction.

The authors also introduce a local refinement technique in which a regular grid structure is used for performing wire length improvement. Cells are rippled by one bin width in one of four directions (up, down, left, or right) based on a gain metric. This metric considers the gain (or loss) in wire length achieved by moving cells in a particular direction, as well as the increase (or decrease) in occupancy of the target bin. If all four scores are negative, the cell remains in its

9 Only the *x*-direction is shown in this example. The *y*-direction is similar.

current bin; otherwise, it moves to the target bin with the highest score [15].

Xiu et. al. introduce the novel concept of "grid-warping" in [58]. In this analytic strategy, quadratic placement is used to initially place the cells. A uniform grid is then superimposed over the placement region. This grid is subsequently deformed (warped) by the "gravity" of the placement "mass" such that individual "bins" in the grid become convex quadrilaterals. An inverse bilinear transform [58] is applied to map the grid shapes back into the original, uniform grid. `GORDIAN`-style partitioning [39] is used to further alleviate congestion—that is, the analytic placement is used to seed a partitioner, which then imposes cutline constraints, effectively dividing the placement region. The combination of partitioning and grid-warping is repeated until cells have been adequately spread.

A new net model for the quadratic placement formulation is presented in [59]. Unlike the clique or star models, the model in [59] is based on Steiner trees built on the Hanan grid (c.f. [60]). These trees are used to construct a quadratic objective geared toward minimizing net delay rather than wire length. The lengths of segments in the trees—and by extension, the representation of the distance between cells—can be individually controlled; thus, this approach gives the net model (and placement engine) more precise control over timing constraints.

2.5 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` [61] uses recursive partitioning to arrive at an initial placement, which is then improved using simulated annealing methods.

Similarly, `GORDIAN` [39], `GORDIAN-L` [44] and `BonnPlace` [34] 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.

Alternatively, an analytic method can use *forces* such that fairly non-overlapping placements are obtained without the need for partitioning. In `Kraftwerk`[10] [13], forces are used to push cells from over-filled toward under-filled regions of the placement area. Mo and Brayton [10, 62] present a similar, force-based strategy but offer an alternative means of calculating forces based on attractive and density-balancing constraints. In [43, 63], dummy cells are used to accomplish cell spreading. In `Mongrel` [64], ripple-moves are used to reduce congestion and improve the quality of placement during the overlap removal stage of global placement [64]; spreading forces are used to direct and control the placer's rippling of cells.

These force-based approaches are of great interest for several reasons. Mixed-size problems—that is, circuits with a combination of standard cells and macro blocks—are handled seamlessly by such heuristics. Furthermore, these methods provide continuous cell locations and therefore appear very amenable to timing- and congestion-driven placement and physical re-synthesis.

10 `Kraftwerk` is the commercial implementation of the ideas presented in [13].

Chapter 3

Force-Directed Global Placement

This chapter begins with a description of how to augment a quadratic formulation with spreading forces to obtain fairly non-overlapping placements. Next, the major implementation details of this thesis' force-directed placer, called FDP, are described. Finally, additional techniques for improving the quality and performance of the heuristic are presented.

3.1 Review of Spreading Forces

Since analytic techniques solve for cell locations while ignoring overlap, additional methods are required to obtain placements that do not violate overlap constraints. In [13], the authors proposed to directly modify the optimality conditions for the quadratic problem in Equation (2.2) via the inclusion of an additional vector of *forces* at each iteration of the placement. The force vector is derived from the distribution of cells throughout the placement region and *perturbs* the optimal solution in (2.2) to remove overlap. The perturbation is selected such that cells are "pushed" away from regions of high density and "pulled" toward regions of low density. That is, at iteration i, the cell locations are determined from the system of equations given by

$$\mathbf{Q}_x\mathbf{x}_i + \mathbf{c}_x + \sum_{l=1}^{i-1} \alpha_l \mathbf{f}_l + \alpha_i \mathbf{f}_i = \mathbf{0} \tag{3.1}$$

where $\mathbf{f}_i$ represents the spreading forces computed at iteration i and α_i represents the weighting with respect to wire length. At each iteration, forces throughout the placement region are computed using an analogy similar to charge attraction or repulsion in an electric force field. Spreading forces are *accumulated* over iterations to avoid placements from "collapsing" back onto themselves. An illustration of placement at various stages of optimization using the quadratic problem in (2.2) with the perturbed optimality conditions given in (3.1) is shown in Figure 3.1.

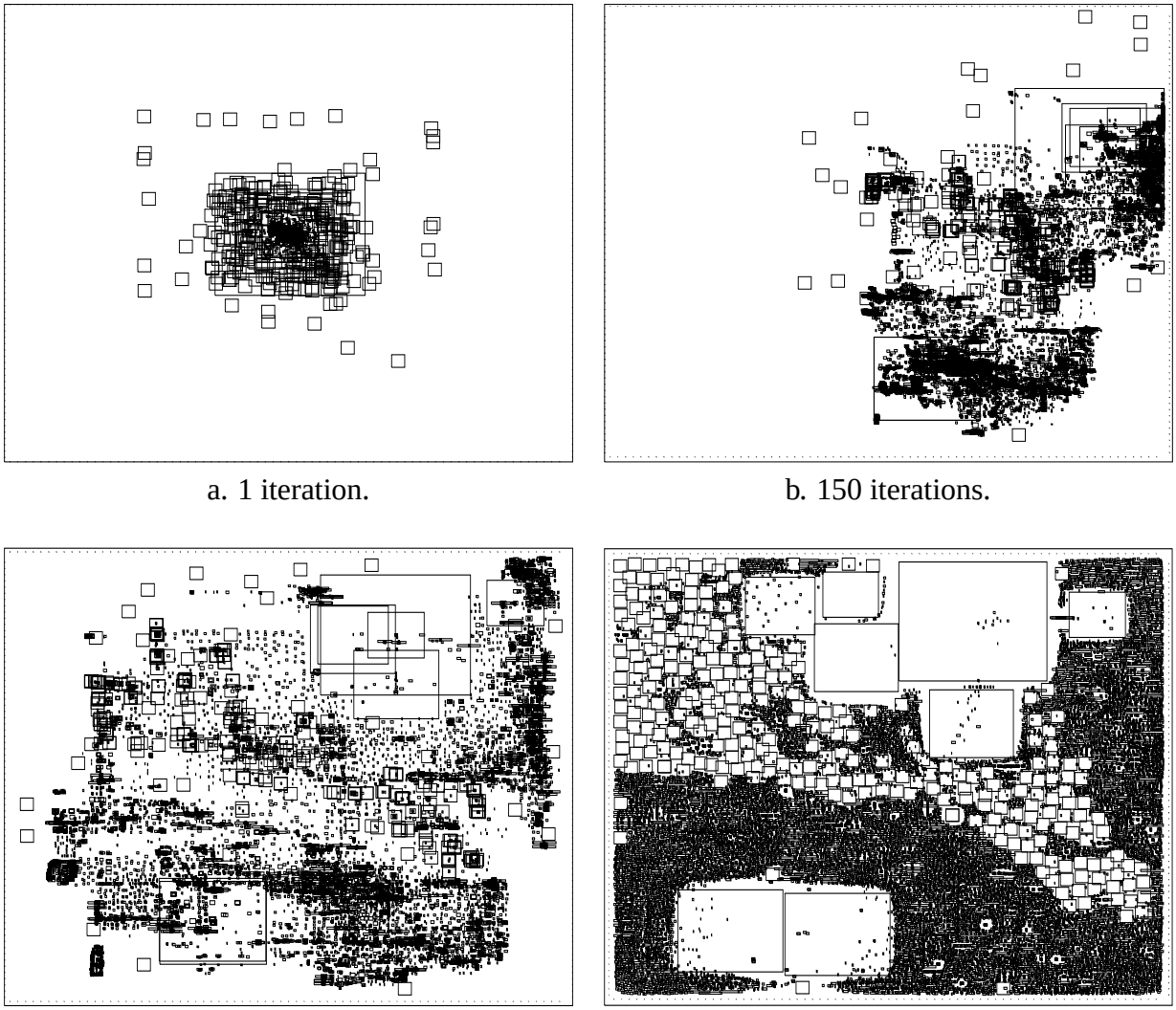
At each iteration i of placement, the vector of spreading forces $\mathbf{f}_i$ is calculated using current cell locations and Poisson's equation given by

$$f(x,y) = k \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} D(x',y') \frac{\vec{r}(x,y) - \vec{r}(x',y')}{|\vec{r}(x,y) - \vec{r}(x',y')|^2} dx'dy' \tag{3.2}$$

where $f(x,y)$ is the force on a cell at location (x,y), $\vec{r}(x,y)$ is the vector representation of the point (x,y) and $D(x,y)$ is the density at point (x,y). $D(x,y)$ represents the ratio of total cell occupancy and allowable capacity at point (x,y); consequently, $D(x,y)$ measures the over-utilization of any point within the placement region. In practice, the force computation is accomplished using discrete bins, with the continuous integration in (3.2) being replaced with discrete summations. Force computation is further elaborated upon in Section 3.2.3, and the selection of force weights is discussed in Section 3.3.2.

3.2 Development of the Core Placement Framework

The development of a force-directed placement framework is by no means trivial. While there are simulated annealing and partitioning-based placers available for free download over the Internet, to the author's knowledge, no analytic placers are available—this may be a testament to the difficulty of developing such a heuristic. The recent literature on force-directed placement (c.f. [10–13]) also fails to address the practical implementation details of this approach; chiefly, how to compute spreading forces, gauge the progress of the placement, and deal with the resultant numerical instability issues that have been observed. The following sub-sections discuss the pertinent details



a. 1 iteration. b. 150 iterations.

c. 220 iterations. d. 310 iterations.

Figure 3.1: Illustration of force-directed placement for a design consisting of a mixture of standard and macro cells. The total number of cells is approximately twenty-seven thousand. Wire length is modeled using the quadratic problem in Equation (2.2) while the perturbed optimality conditions in Equation (3.1) are used to determine cell locations to simultaneously account for both wire length and cell overlap.

surrounding the engineering of FDP.

An outline of the CAD flow implemented by FDP is shown in Figure 3.2. FDP begins by parsing a netlist file in either of the two commonly-accepted academic formats, Bookshelf [65] or VPR [7]. The circuit is then passed to the force-directed global placement phase.

The general flow of the global placement phase is shown in Figure 3.3. The heuristic begins by constructing a set of Hessian matrices for the *x*- and *y*-directions. The forces are initially set to zero, and an unperturbed quadratic placement is solved using a conjugate gradient technique (discussed in Section 3.2.1). Based on the amount of spreading, forces are calculated (see Section 3.2.3), scaled (see Section 3.3.2), and added to the quadratic formulation (as in Section 3.1). This updated problem formulation is then solved, and the algorithm is repeated. If, after solving the quadratic problem and updating cell positions, it is found that the problem destabilized, then *friction* is applied, as discussed in Section 3.2.5, and the algorithm continues. Once a circuit is sufficiently well-spread (see Section 3.2.4), the placement framework proceeds to legalization (see Section 4.1). Once legalized, the locations of each cell in the netlist are written to an output file.

3.2.1 Matrices and Linear Solver

The matrix library and solver lie at the heart of a force-directed placer. In developing FDP, various open-source C++-based linear system packages were investigated. The MTL library [66] was initially chosen for sparse matrix support due to its tight coupling with the well-known iterative library, ITL [67]. However, recent experimental results comparing MTL to the Boost uBLAS matrix library [68] have found uBLAS to be more efficient in matrix-vector operations [69]. Consequently, uBLAS was adopted as the underlying sparse matrix package for the placer. The generalized "vector of vectors" compressed storage format is used for the matrices, as this has been found to offer both good performance and reasonable memory consumption.

As there were no free linear solvers available for uBLAS, a new solver had to be written. Based on experiments and the results described in [70], a stable bi-conjugate gradient (BiCGStab) solver [71] with an ILU(0) preconditioner [72] was developed. To further improve performance,

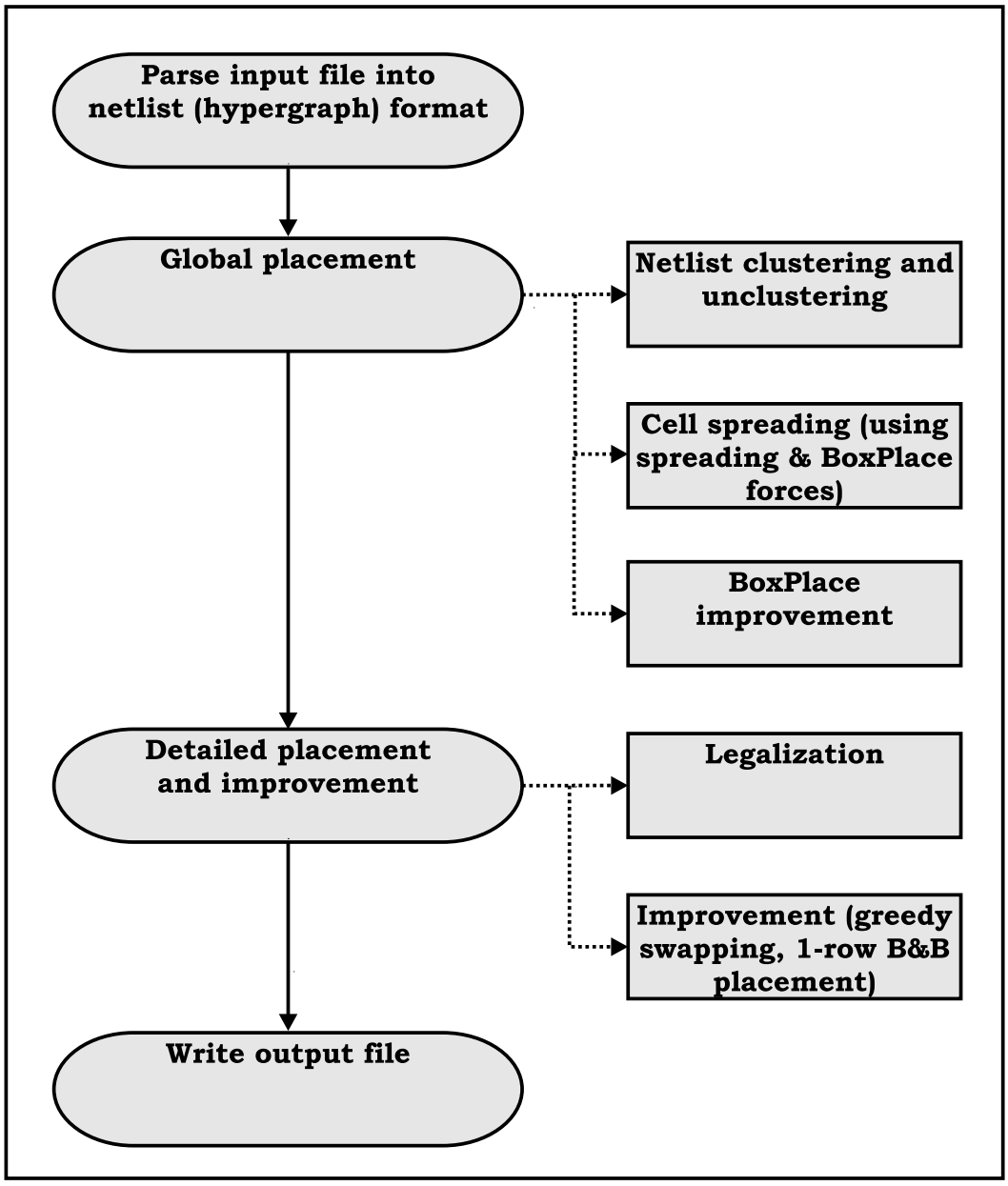


Figure 3.2: Diagram of the flow in the analytic placement framework.

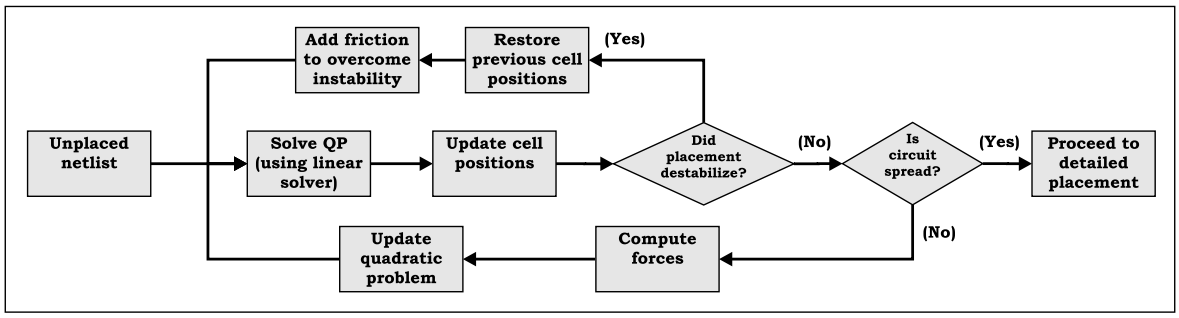


Figure 3.3: Diagram of the general flow of the force-directed placement.

the Reverse Cuthill-Mckee [73] algorithm was implemented to reorder the Hessians.

3.2.2 Net Models

A hybrid clique/star model has been found to slightly improve solver performance and reduce memory consumption. This performance improvement is accomplished through a reduction in the number of non-zero entries in the Hessians, which renders the matrices more diagonalized and solvable more quickly using a conjugate gradient technique. The findings of [15] may lead one to conclude that drastic performance improvements can be achieved using a hybrid model. While the importance of using this model to mitigate the quadratic edge count of large clique nets is undeniable, the use of a star model for 4-pin (and greater) nets imparts only a slight (less than 10%) performance gain, at the expense of 1%–2% in wire length. Consequently, when constructing the Hessians, FDP uses a clique model for nets smaller than 20 and a star model for all larger nets, as this has been observed to produce better quality placements than when using the sizing advocated by [15].

Moreover, when using this hybrid net model, a clique (and star) net weighting of $\frac{1}{k-1}$ (and $\frac{k}{k-1}$), for nets with k pins, has been observed to produce the best results. This observation agrees with the net weighting used in [15]. The weighting of $\frac{1}{(k-1)^2}$, as suggested in [74], was tested, but the resultant placements were found to be 3%–5% worse on average.

3.2.3 Force Computation

Spreading forces are computed using a Barnes-Hut quad-tree for *n*-body force calculation [75]. This method was chosen for its good overall performance and its relatively high precision [76]. High accuracy (when calculating forces) was found to be especially important in the early stages of placement, and it is for this reason that this algorithm was employed in FDP. Since the quad-tree tracks cell locations and the distribution of cell area, it offers the additional advantage that it can be employed in other algorithms, as discussed in Section 3.2.4.

In this approach, a quad-tree is constructed over the placement area such that, at the lowest level of the tree, each bin contains approximately one cell (see Figure 3.4). Therefore, the quad-tree is built bottom-up, and the bins in the lowest level of the tree are sized by computing the average width and height of the cells. Given a current placement, cell area is inserted into all levels of the quad-tree based on cell location. Then, for each bin in the bottom level of the tree, the forces acting on the bin are accumulated (summed) using interaction lists and near neighbour computations. Finally, the actual force on each individual cell is computed by summing the bin forces from those bins that the cell overlaps.

Large cells may span several quad-tree bins, and may receive contributions to their forces from the many bins that they occupy. As a result, the largest cells tend to receive the largest forces. For placements with a wide distribution of cell dimensions, such as the mixed-size benchmarks introduced in Chapter 4, the relative magnitudes of forces for different cells can vary considerably—tests have suggested that this can cause the largest cells to spread too quickly compared to the smaller cells, harming wire length. Dividing the magnitude of the forces on each cell by the square root of the number of bins that it occupies (at each level in the quad-tree) has been found to lead to a more even distribution of force magnitudes, as shown in Figure 3.5. This empirically results in better placements.[1]

1 Force weights also play an important role in spreading the cells and preserving quality, and this issue is addressed in Section 3.3.2.

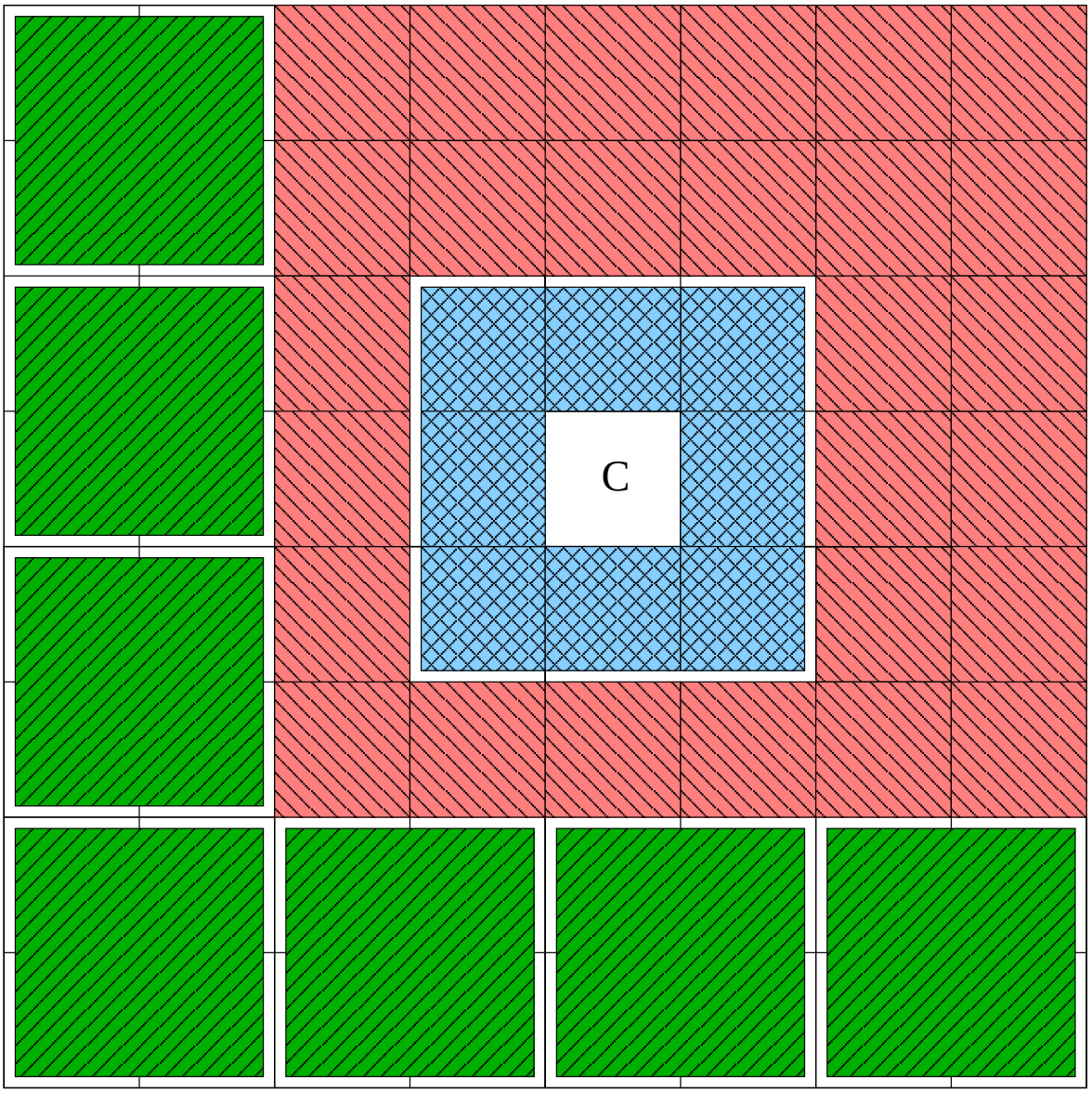
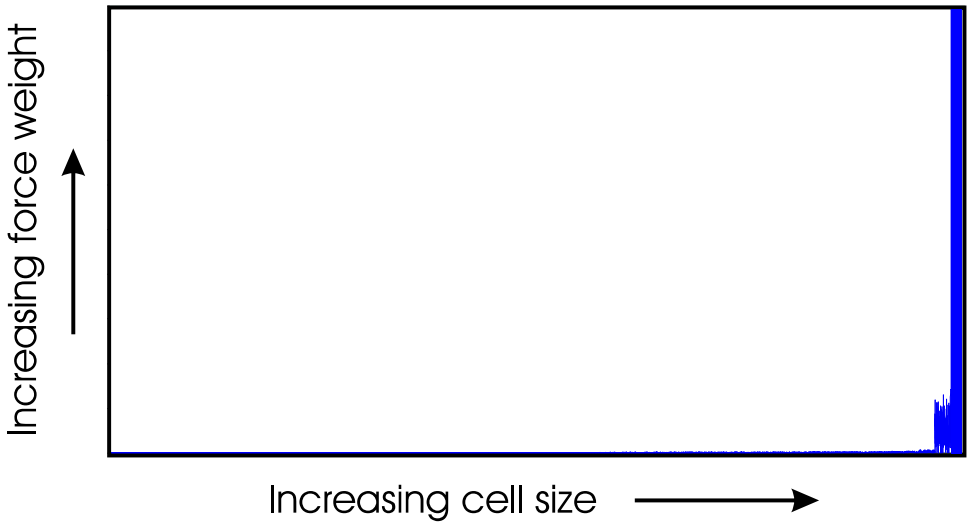
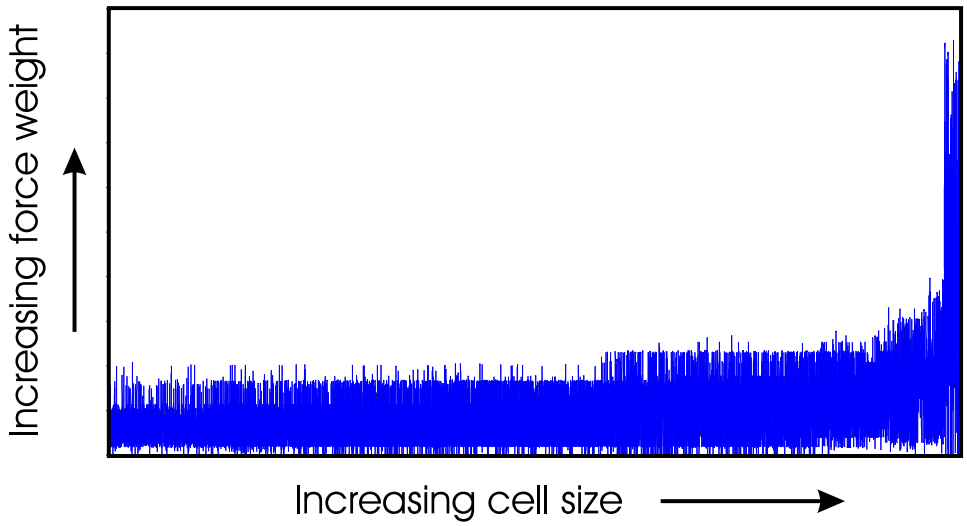


Figure 3.4: Diagram of the bin levels in the Barnes-Hut quad-tree. This diagram shows a bin C, at the lowest grid level of the force tree. Forces are computed between the bin and those bins which are in C's interaction list. Then, the contributions from the near neighbours are summed. In this example, there are only two interaction lists—the first, at the third quad-tree level, possesses 27 bins, and the other, at the second quad-tree level, possesses 7 bins.



a. Before force scaling



b. After force scaling

Figure 3.5: Distribution of force magnitudes before and after scaling by the square root of the number of quad-tree bins. The scales on the axes are linear in the number of cells and the size of the force weights. Note that prior to scaling, forces on large cells are significant whereas the forces on small cells are almost negligible (barely visible in the diagram). After scaling, forces are more uniform, although larger cells still receive justifiably larger forces.

3.2.4 Spread Metrics and Stopping Criteria

In force-directed placement, there is a need to accurately measure the progress of the placer in minimizing cell overlap. In the absence of an adequate *spread metric,* it is difficult to assess how much better (or worse) one placement is when compared to the placement of a previous iteration. Moreover, it is difficult to judge whether or not the spreading forces are properly weighted, or to determine when to terminate the placement and continue with legalization. To this end, two metrics were developed to assess the spreading of cells.

The first metric is based on the violation of cell density as it occurs in the Barnes-Hut quad-tree. The metric is calculated by traversing the quad-tree using Breadth-First Search (BFS) and adding the factor by which the occupancy of a geometric region exceeds its allowable capacity, scaled by the square of the quad-tree level; that is,

$$\sum_{l=1}^{L} \frac{1}{l^2} \sum_{i,j:\mathrm{occ}(i,j)>\mathrm{cap}(i,j)}^{2^l} \frac{\mathrm{occupancy}(i,j)}{\mathrm{capacity}(i,j)}$$

is computed for bins (i, j) at each level l of the tree. This weighting places more emphasis on capacity violations in the top levels of the quad-tree. Given that the worst possible value would occur when all cell area is focused on the smallest bin at each level of the quad-tree, this spread metric can be normalized to within the interval $[0,1]$ with a metric closer to 0 implying less over-utilization of available placement area.

A second spread metric, based on Klee's measure problem [76, 77], is also employed. The $O(n \log n)$ segment tree technique in [77] was implemented to measure the area of the union of the modules in the placement region. This area is then divided by the total cell area to give a normalized value in the interval $[0,1]$. The resulting value measures the percentage overlap remaining in the placement, with a metric closer to 1 implying less overlap between cells. An example of the calculation is shown in Figure 3.6.

Each of the spread metrics offers a different insight into the amount of cell spreading throughout the placement area. The first metric, based on the violation in the quad-tree, works

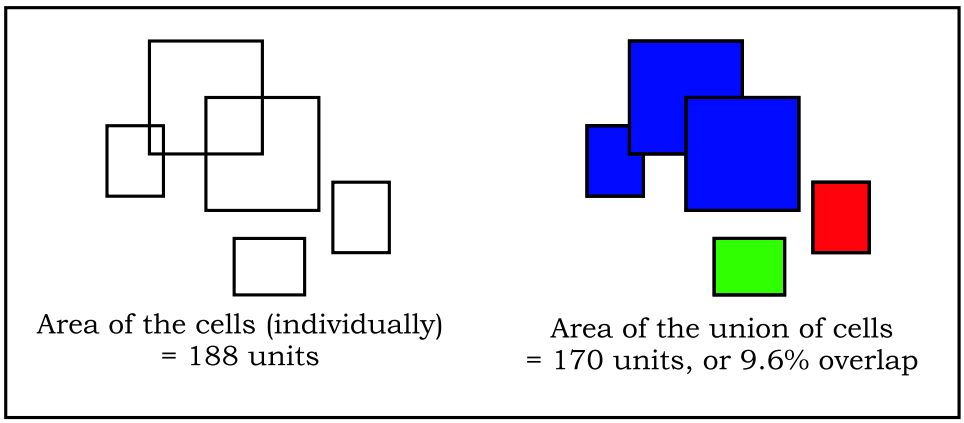


Figure 3.6: Sample Klee's measure computation for overlap calculation. The area of the union of the cells is divided by the area of the cells to yield a metric representative of the amount of overlap in the placement.

well during the early stages of placement. However, near the end of placement, when cells are well-spread, small changes in overlap are not accurately reflected in this metric. Klee's measure technique, on the other hand, offers little insight into the quality of spreading early in the design, when there is significant cell overlap. However, it provides a very detailed view in the final iterations of placement, when it becomes important to measure minute improvements in spreading. Consequently, it has been observed that a 50%/50% combination of the two metrics works best in assessing the amount of overlap at any iteration of the placement flow.

Unlike the approach used by [13], in which global placement is stopped once there are "no empty squares within the placement area which is larger than four times the average area of a cell", the spread metrics described here can be used to stop placement in designs with large amounts of whitespace. Experiments have shown that placement can finish when there is 30%–35% overlap remaining, as computed by Klee's measure. As noted in [64], the force-directed approach can require *many* iterations to completely purge overlap from a circuit. Consequently, this stopping point offers a good trade-off between placer performance and overlap removal. It is computationally more efficient to employ another algorithm as part of an intelligent legalization scheme to remove residual overlap.

Numerical results substantiating this choice of stopping point are provided in Table 3.1. In

this experiment, placement was stopped at varying degrees of overlap, with the run-times and wire lengths reported for both pre- and post-legalized circuits. These results confirm that stopping a force-directed placer with 30%–35% overlap remaining offers a good mix of performance and quality.

3.2.5 Friction and Stability

It was discovered that the incremental addition of a small amount of spreading forces from one placement iteration to another could result in a "destabilization" of the placement. In these cases, initially-spread cells could "collapse" into the edge of the placement region. This phenomenon is illustrated in Figure 3.7, and is easily detected by monitoring the aforementioned spread metrics. Investigations indicated that this placement destabilization occurred due to the ill-conditioning of the Hessians. A method with a physical interpretation was sought to avoid the "destabilization" of the placement.

Recall that the cell locations at iteration i (x-direction only) are given by

$$\mathbf{Q}_x\mathbf{x} + \mathbf{c}_x + \mathbf{f}_x = \mathbf{0} \tag{3.3}$$

while at iteration $i+1$, cell locations are given by

$$\mathbf{Q}_x\mathbf{x}' + \mathbf{c}_x + (\mathbf{f}_x + \Delta\mathbf{f}_x) = \mathbf{0} \tag{3.4}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are the cell locations at iteration i and $i+1$, respectively. Subtracting (3.3) from (3.4) yields

$$\mathbf{Q}_x(\Delta\mathbf{x}) + \Delta\mathbf{f}_x = \mathbf{0} \tag{3.5}$$

where $\Delta\mathbf{x} = \mathbf{x}' - \mathbf{x}$. In physical terms, (3.5) indicates that the increment in cell locations, $\Delta\mathbf{x}$, is a function of the incremental force and the interconnections between movable cells. Hence, if $\mathbf{Q}_x$ is poorly conditioned, then small changes to the spreading forces, $\Delta\mathbf{f}_x$, could result in large changes

Table 3.1: Results comparing legalized wire length when using different overlap stopping points for a standard cell placement instance with approximately thirteen thousand cells. HPWL values have been divided by 10^6.

Overlap (%)	**Number of Placement Passes (CPU Time)**	**Pre-Legal HPWL**	**Post-Legal HPWL**
50	243 (181 s)	1.48	1.76
45	287 (233 s)	1.54	1.74
40	310 (257 s)	1.57	1.73
35	345 (296 s)	1.61	1.71
30	374 (325 s)	1.65	1.71
25	390 (353 s)	1.67	1.70
20	410 (377 s)	1.70	1.70

to the cell locations since $\|\Delta\mathbf{x}\| = \|\mathbf{Q}_x^{-1}\Delta\mathbf{f}_x\| = \|\mathbf{Q}_x^{-1}\|\|\Delta\mathbf{f}_x\|$.

To improve conditioning, the system of equations was altered by using a subset of "virtual" fixed points (i.e., dummy cells without height or width) positioned at the locations of a subset of movable cells in the netlist. Each added virtual fixed point was connected to its corresponding movable cell via a weighted virtual edge. This change only affects the diagonals of the Hessian, while improving conditioning by making $\mathbf{Q}_x$ more diagonally dominant. With the addition of these virtual fixed points and weighted virtual edges, (3.5) becomes

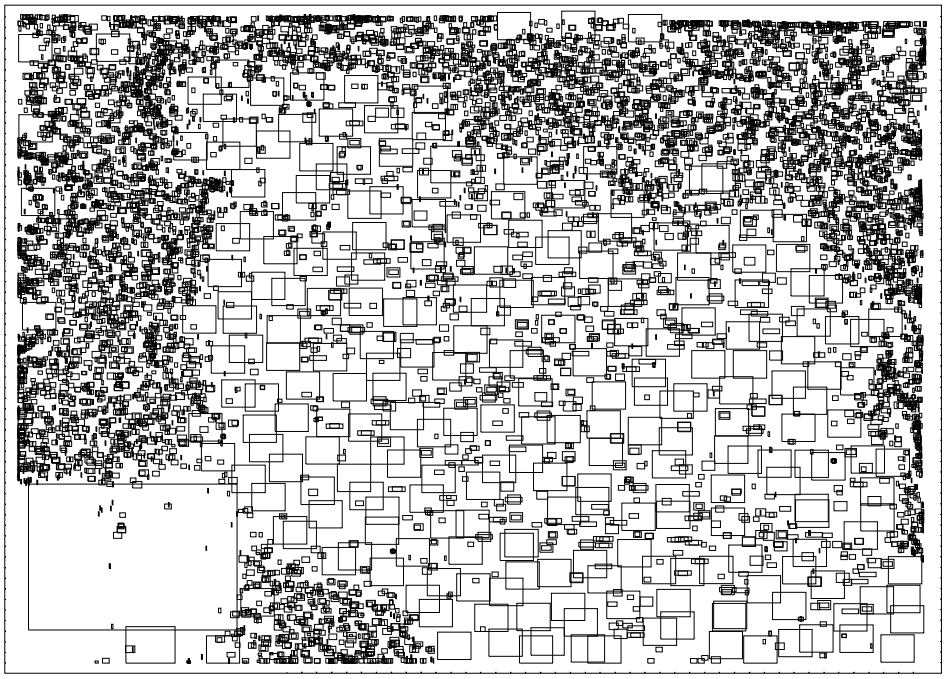
$$\mathbf{Q}_x(\mathbf{x}' - \mathbf{x}) + \mathbf{w}\mathbf{I}(\mathbf{x}' - \mathbf{x}) + \Delta\mathbf{f}_x = \mathbf{0}$$

which yields

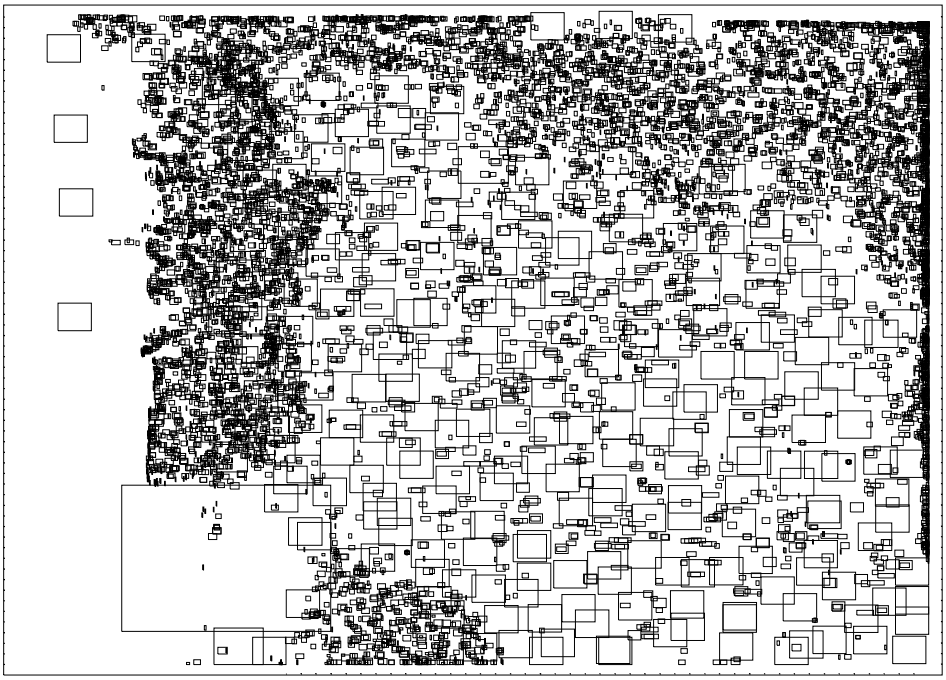
$$\Delta\mathbf{x} = -(\mathbf{Q}_x + \mathbf{w}\mathbf{I})^{-1}\Delta\mathbf{f}_x \tag{3.6}$$

where the weighted identity matrix $\mathbf{w}\mathbf{I}$ is due to the weighted connections to the virtual fixed points. Since the diagonals will be larger, (3.6) has the effect of also being easier and faster to solve. Moreover, if one matrix dominates the other ($\mathbf{w}\mathbf{I} \gg \mathbf{Q}_x$), the system of equations can be approximated by

$$\Delta\mathbf{x} \approx (\mathbf{w}\mathbf{I}^{-1})\Delta\mathbf{f}_x \approx \mathbf{w}^{-1}\Delta\mathbf{f}_x. \tag{3.7}$$



a. Before destabilization



b. After destabilization

Figure 3.7: Illustration of destabilization during placement for a circuit with thirteen thousand cells. Note how the cells have shifted toward the right of the placement area. This destabilization can be detected using the spread metrics by checking for large (greater than 5%) increases in the amount of overlap.

Equation (3.7) reveals that, if a sufficient number of virtual fixed cells are added at the locations of the movable cells, then the step that will be taken will be in the direction of the incremental force, and that only some fraction (controlled by **w**) of that distance will be traveled. Physically, this weighted step in the direction of the spreading forces can be interpreted as *friction*.

Experiments conducted with friction have indicated that only a small percentage of the rows in the Hessian—usually, only around 5% of the least diagonally-dominant rows—need to have virtual fixed cells attached to improve conditioning and stability. After friction is applied and the previous iteration's cell locations are restored, subsequent placement iterations usually do not destabilize. Moreover, the use of fixed virtual cells does not degrade the placement results relative to the results obtained without the application of friction.

3.3 Improvements to the Core Placement Framework

A force-directed placer based on the concepts presented in Section 3.2 was implemented. Yet, experiments performed with this early placer yielded results that were 5% to 10% worse than `Kraftwerk`. Numerous additional techniques were thus required to bring the quality of results to a level comparable with other modern tools, like `Capo` [32] and `mPG` [14].

BoxPlace, described in Section 3.3.1, is this dissertation's primary quality-enhancing technique. BoxPlace augments the global placer in two ways: it directly repositions cells to minimize HPWL, and it is used to reorient spreading forces to point in a direction that not only encourages spreading, but also favours wire length minimization. Using BoxPlace in these two manners has been observed to improve wire length by 5%–10%.

In Section 3.3.2, a dynamic force weighting schedule is presented. This schedule modifies the force scalar α_i in Equation (3.1) to modify the rate at which cells spread. This dynamic weighting affords better control over the increase in wire length and in the rate of spreading of cells than does a static weighting scheme.

Lastly, Section 3.3.3 describes the importance and the implementation of a Hybrid First Choice clustering scheme. A small amount of clustering was found to improve wire length by

preventing spreading forces from occasionally pushing cells—which should otherwise remain close together—too far apart.

3.3.1 BoxPlace

Foremost among this thesis' quality-enhancing methods is a technique called "BoxPlace" [47]. BoxPlace moves each cell to the median location of its connected nets, thereby reducing wire length directly. The pseudocode for this algorithm is shown in Figure 3.8.

In BoxPlace, the range of (x,y) values which minimize HPWL for the nets connected to cell *i* is calculated as follows. First, the *x* and *y* minima and maxima of the bounding boxes for all nets (excluding the points contributed by cell *i*) are inserted into two vectors—one for each direction. The vectors are then sorted by increasing value and the median locations—the locations ($\lfloor n/2 \rfloor$) and ($\lfloor n/2 \rfloor + 1$) in both vectors—yield the "box" (or range) of locations into which cell *i* can be moved to improve wire length. By using a grid-based structure, cell locations can be tracked to ensure that they are inserted into a relatively under-occupied area within their target box. Tracking cell locations and cell area to prevent the re-introduction of cell overlap is an enhancement of the BoxPlace algorithm described in [47]. With respect to FDP's placement flow, calls to BoxPlace are performed after every 3% reduction in cell overlap as measured by the spread metrics. Since BoxPlace can *reintroduce* overlap, cell spreading is re-evaluated after each pass of BoxPlace—if too much overlap is introduced, the previous cell locations are restored, and the algorithm stops.

BoxPlace is also used to compute "minimizing forces" which are employed in conjunction with the spreading forces and the quadratic wire length objective. These minimizing forces are simply vectors which point to the locations to which cells should move to reduce HPWL, as computed by the BoxPlace algorithm. These forces are scaled appropriately, and then combined linearly with the spreading forces described in Section 3.2.3.

Experiments have shown that the combination of BoxPlace and spreading forces achieves the most favourable quality and performance trade-off when combined in a 40%/60% ratio. This vector addition reorients the angles of the spreading forces to point in a direction that *favours* spreading,

```
1   Procedure: BOXPLACE
2   Input: A netlist, N
3   begin
4     E ← all eligible, movable cells in N;
5     for each pass ∈ 0,1,...,MAX_PASSES do
6       lastPosn ← location of all cells;
7       Randomly permute E;
8       for each n_i ∈ E do
9         Find the range ("box") of (x,y) values that minimizes HPWL for n_i;
10        Move n_i to a relatively non-overlapping area somewhere within the "box";
11      od
12      if too much overlap was reintroduced then
13        Restore the cell locations from lastPosn;
14        break ;
15      fi
16    od
17  end
```

Figure 3.8: Pseudocode for the BoxPlace algorithm.

but additionally minimizes wire length. This modification to the spreading forces is illustrated in Figure 3.9. A comparison of the wire lengths versus run times for different combinations of BoxPlace and spreading forces is provided in Table 3.2.

BoxPlace, when employed individually and when used to modify the direction of the spreading forces, can significantly improve wire length. It helps to achieve a more linearized measure of wire length without degrading performance as much as a linearized re-weighting scheme, such as the one previously mentioned in Section 2.4.2. BoxPlace forces have been found to be especially important in improving the quality of placement above and beyond that of `Kraftwerk`, as discussed in Chapter 4.

Finally, BoxPlace aids cell spreading in another, subtler fashion. One of the difficulties with achieving high-quality placements lies in the fact that forces do not allow cells to "flip" sides—once a cell is located to the left or right of another cell, the spreading forces will not allow the two to cross paths, even if doing so would improve wire length. This problem can be particularly troublesome given that relative cell ordering is often established early on in placement (after the initial quadratic problem is solved). This is when there exists the greatest amount of "uncertainty"

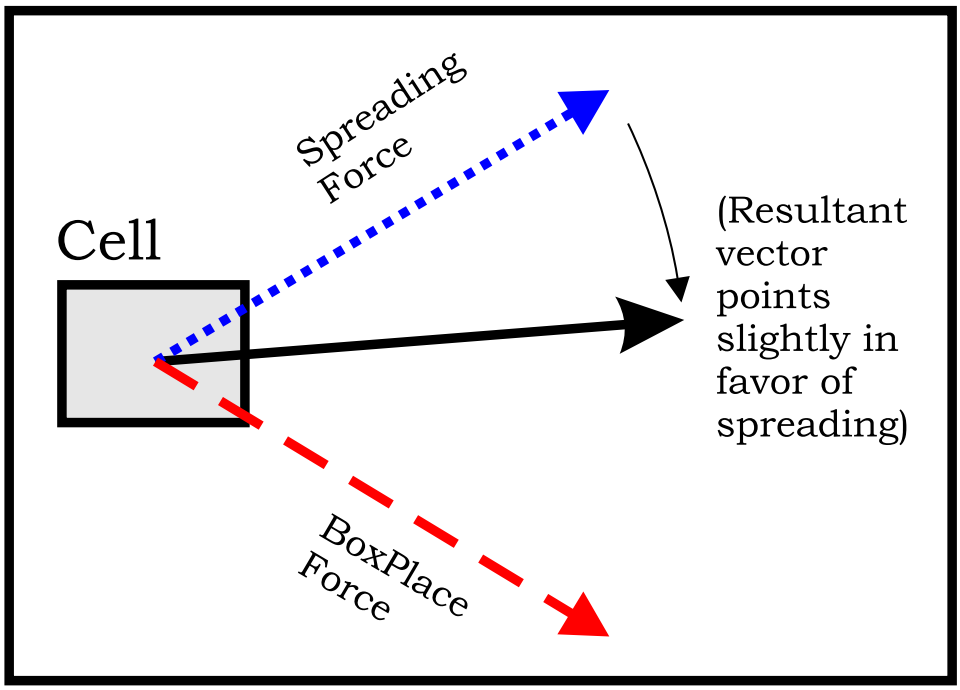


Figure 3.9: Illustration of the addition of spreading and BoxPlace forces, and the resultant vector.

in the placement due to cell overlap, and when an incorrect amount of spreading force is most likely to do the greatest "damage" to the wire length. BoxPlace, on the other hand, performs this "flipping" and situates cells in more favorable locations to reduce wire length.

3.3.2 Dynamic Force Weighting

The weights applied to the spreading forces impact the speed at which a circuit spreads, and can affect the quality of the overall wire length [13, 64]. Experiments have shown that *weighting forces via a constant, as advocated by [13], leads to poor placements*.

In FDP, the force weight is adapted *dynamically* in each iteration to achieve both fast spreading and good quality. First, the force vectors for each iteration are normalized with respect to the largest force vector. Then, the force weighting schedule weights the elements of $\mathbf{f}_x$ and $\mathbf{f}_y$ by a scalar value α. The value of α in each iteration is adjusted based on the following experimental observations:

Table 3.2: Results comparing different combinations of BoxPlace and spreading forces for a mixed-size placement instance with approximately thirteen thousand cells, at 33% overlap. The use of BoxPlace forces in the ratio of 40%/60% has been observed to offer the best quality/performance trade-off.

BoxPlace (%)	**Spreading (%)**	**Number of Passes (CPU Time)**	**HPWL ($\times 10^6$)**
≥ 50	Placement Did Not Complete		
40	60	271 (380 s)	2.25
30	70	250 (362 s)	2.29
20	80	245 (341 s)	2.33
10	90	231 (327 s)	2.30
0	100	222 (321 s)	2.29

- Initially, the force weight α should be as small as possible. While good spreading can still be achieved if large force weights were used in the first 10 to 50 iterations of placement, wire length may be significantly worsened.

- Once the relative ordering of cells has been well established, and the circuit begins to spread consistently, α can be increased in each iteration to encourage faster spreading. α should be reduced if the spread metrics detect too much reduction in the amount of overlap (and vice-versa for too little improvement in the spreading).

In other words, force weights are adapted continuously using, in effect, a three-step state machine in which the spread metrics are used to help in dynamically weighting forces—if there is too much spreading, force weights are lowered, and if there is too little, force weights are raised. The spread metrics can also help to determine if the placement is "oscillating" (no longer spreading), in which case FDP can stop and proceed to legalization.

It should be further noted that to achieve consistent force weights across a variety of circuits, the placement region (and all cells) are scaled by the average cell height and width. In general, this scaling does a good job of normalizing the force weighting such that different sizes of placement regions are affected in roughly the same way by the same force weight.

3.3.3 Clustering

The final technique for improving quality is the application of a small amount of Hybrid First Choice clustering [29], which can improve the wire length of a placement by up to 3%. Clustering the netlist tends to "smooth" the differences in cell heights and widths. The more the cells are similarly-sized in a problem, the less negative impact that spreading forces tend to impart on the cells. This is due mostly to the fact that large cells tend to possess considerable overlap in the early stages of placement, and may receive large "pushing" forces early on—these large forces can negatively impact wire length by moving large cells to sub-optimal positions too quickly. Thus, clustering ensures that nodes with high affinities remain close together throughout the placement.

In Hybrid First Choice clustering, cells are initially placed onto a "free" list which contains the set of cells which have not been paired. The *affinity* for pairing any node i with any node j is then calculated using the following formula:

$$r_{ij} = \sum_{e \in E_h \,|\, i,j \in e} \frac{1}{|e| - 1}. \tag{3.8}$$

The algorithm repeatedly removes the node with the highest affinity from the free list, and pairs it with the node that (originally) yielded this high affinity, even if that node had already been paired. Once a node has been paired, it is said to form a "cluster". In this algorithm, an unpaired node is always paired with either another unpaired node or a cluster.

It is important to note that a limit must be placed on both the number and size of clusters or else the resulting clustered netlist will consist of too few, overly-large blocks (and placement quality will suffer as a result). To this end, it suffices to *not* match a node i with a cluster j if the aggregate area of the pairing (the sum of the area of node i and all of the paired nodes contained within the cluster) would exceed four times the average cell area of the original netlist. In this circumstance, cluster j would be avoided, and node i would simply be paired with its next best affinity match which satisfies this area constraint. The affinity calculation in Equation (3.8) could alternately be

modified along the lines of [78] to encourage more matchings between smaller cells:

$$r_{ij} = \sum_{e \in E_h \mid i,j \in e} \frac{1}{(|e|-1) \cdot \text{area}(e)}.$$

where area(e) represents the area of all nodes (excluding node *i*) on edge *e*. In addition, nodes are only allowed to be paired if their affinity is greater than or equal to the median affinity of all possible pairings—this encourages the algorithm to accept only the best possible pairings.[2]

2 This restriction allows only up to 50% of the original netlist to be paired in one pass of the algorithm.

Chapter 4

Results and Analysis

In this chapter, the strategy for detailed placement is described, and results for FDP are compared with other leading academic tools. While FDP can place FPGA, standard cell, *and* mixed-size designs, this thesis focuses primarily on the latter two, by way of Bookshelf-format circuits [65].

Standard cell benchmarks are attractive because they include large test circuits which allow for easy comparison of the performance curves of competing placement heuristics. In contrast, the largest academic FPGA circuit is less than $\frac{1}{25}^{\text{th}}$ the size of the largest standard cell test circuit. Furthermore, standard cell and mixed-size benchmarks feature a wide variety of cell dimensions, whereas academic FPGA circuits possess only unit-sized cells. Most importantly, the Bookshelf circuit format is supported by a wide range of placement software, including annealers (e.g., Dragon [61]), partitioners (e.g., Capo [35]), and analytic or hybrid placers (e.g., mPG [79] and Kraftwerk [13]). On the other hand, only one other academic placer—VPR—is presently capable of placing FPGA circuits. Therefore, FPGA benchmarks would do little to facilitate run-time or quality comparisons between various placement heuristics. It is for these reasons that the focus remains on standard cell and mixed-size problems in the following sections.

4.1 Detailed Placement for Standard Cell and Mixed-Size Designs

Placements produced by FDP are not "valid" in that cells are not assigned to rows and some residual overlap may still be present. FDP must be used in conjunction with a legalizer in order to produce non-overlapping placements.

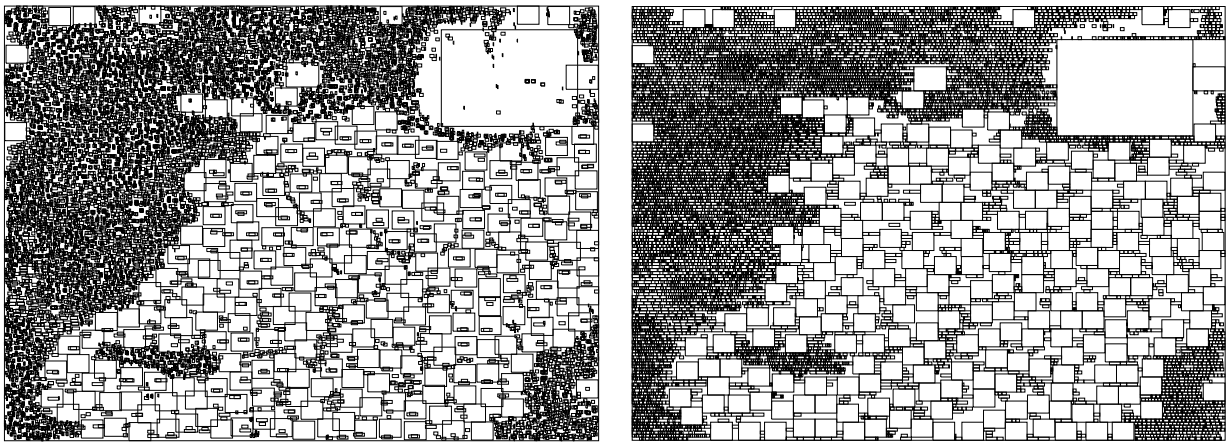
The standard cell legalization strategy adopted in FDP is presently quite simple. Cells are initially snapped to their closest rows in order to minimize total cell movement. Greedy juggling of cells is performed between rows to meet width restrictions imposed by the fixed die. Finally, same-size cell swaps and single-row branch and bound arrangement are applied on groups of cells [22].

In the case of circuits containing macro cells, the placement is first tested using a sequence pair analysis along the lines of [80]. Generally, macro cells fit and no additional work is required. Of course, operations on the sequence pair could be applied, as in [80], if the macro cells do not fit inside the fixed die. Using the results of the sequence pair analysis, macro cells are shifted to align with rows and remove overlap in the y-direction, and shifted to the left and right to remove overlap in the x-direction. Currently, no attempt is made to optimize whitespace in the final placement. Examples of two placements before and after legalization are shown in Figure 4.1.

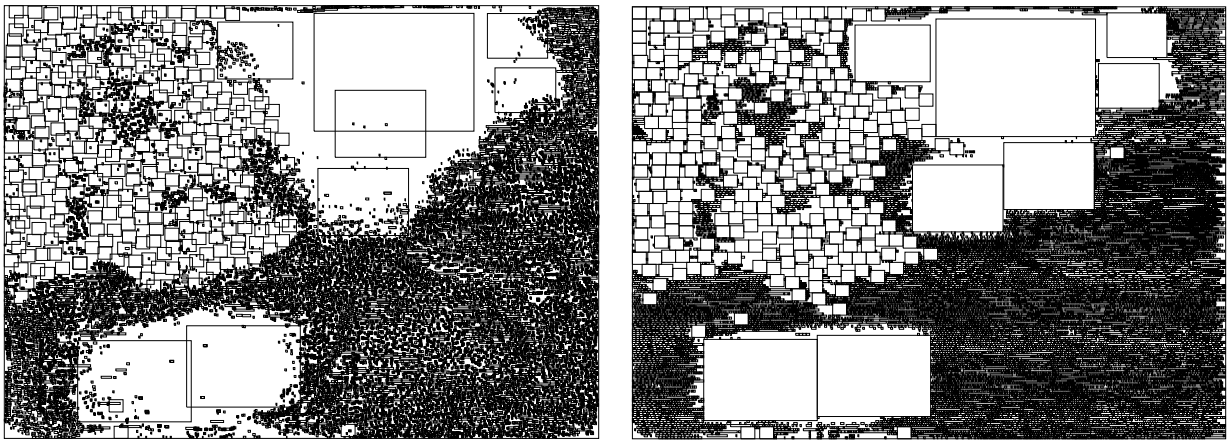
4.2 Testing Methodology

In this section, FDP is compared to a number of other academic placement tools on a variety of different problems, including standard cell and mixed-size designs that contain a number of macro cells in addition to a large number of standard cells. In cases where placement tools were run by the author, software runs were conducted on a Pentium 4, 2.8 GHz machine with 1 GB of RAM running Fedora Linux. Run times are expressed in minutes where applicable. Wire lengths are reported using HPWL divided by 10^6.

The ISPD2002 benchmark circuits from [81–83] are used for testing. These circuits are large-scale, mixed macro and standard cell circuits with the exception of ibm05, which does not have any



a. Circuit with 13 k cells before and after legalization.



b. Circuit with 27 k cells before and after legalization.

Figure 4.1: Illustration of two circuits before and after legalization.

macro cells. The statistics for these circuits are shown in Table 4.1. In the table, column 1 gives the benchmark name. Columns 2 through 5 provide the statistics for the benchmark, including the number of standard cells, macro cells, I/O pads, and hyperedges, respectively. The total area of macro cells versus the total area of standard cells expressed as a percentage is provided in column 6. The area of the largest macro cell versus the total area of standard cells in provided in column 7. Finally, the ratio between the largest macro cell, the smallest macro cell, and the smallest standard cell is provided in column 8. These circuits have been modified to give an aspect ratio close to 1.0, with 20% whitespace available.

For standard cell test circuits, the mixed-size designs were modified by reducing the macro cells to standard cell dimensions. In these designs, the standard cell height is 16 units. Hence, for standard cell tests, all macro cells were reduced to be cells of 16×16 units. The available whitespace in the designs was also adjusted to be 5%. Finally, the placement area was modified to maintain an aspect ratio close to 1.0, which is typical for modern chip design.

4.3 Numerical Results

4.3.1 Comparisons to Kraftwerk

In the first set of experiments, the quality of placements produced by `FDP` *prior* to legalization were compared to those placements produced by `Kraftwerk` [13]. Upon termination, `Kraftwerk` was found to produce placements with approximately 30% overlap as measured by the Klee's measure technique; thus, `FDP` was configured to terminate with approximately the same amount of cell overlap. The results of these tests are presented in Table 4.2. Compared to `Kraftwerk`, `FDP` achieved wire lengths that are, on average, 7% better, thereby validating the efficacy of the methods presented in this thesis.

Table 4.1: ISPD2002 mixed-size test circuits. These circuits are large-scale, mixed designs with combinations of macro and standard cells (with the exception of `ibm05`, which does not have macro cells).

Circuit	**Cells**	**Macros**	**Pads**	**Nets**	A_m	A_m^b	$A_m^b/A_m^s/A_c^s$
ibm01	12260	246	246	14111	67.13%	6.37%	8416:252:1
ibm02	19071	271	259	19584	76.89%	11.36%	30042:240:1
ibm03	22563	290	283	27401	70.75%	10.76%	33088:240:1
ibm04	26925	295	287	31970	59.82%	9.16%	26593:240:1
ibm05	28146	0	1201	28446	0.00%	0.00%	-
ibm06	32154	178	166	34826	72.90%	13.64%	36347:175:1
ibm07	45348	291	287	48117	52.56%	4.75%	17578:240:1
ibm08	50722	301	286	50513	67.35%	12.11%	50880:240:1
ibm09	52857	253	285	60902	52.42%	5.42%	29707:240:1
ibm10	67899	786	744	75196	81.37%	4.80%	71299:252:1
ibm11	69779	373	406	81454	49.76%	4.48%	29707:240:1
ibm12	69788	651	637	77240	73.00%	6.43%	74256:252:1
ibm13	83285	424	490	99666	47.64%	4.22%	33088:240:1
ibm14	146474	614	517	152772	26.72%	1.99%	17860:144:1
ibm15	160794	393	383	186608	43.34%	11.00%	125562:240:1
ibm16	182522	458	504	190048	48.71%	1.89%	31093:252:1
ibm17	183992	760	743	189581	23.78%	0.94%	12441:252:1
ibm18	210056	285	272	201920	11.96%	0.95%	10152:243:1

Table 4.2: Standard cell benchmarks comparing `Kraftwerk` to `FDP` at approximately the same amount of pre-legal overlap. The `Kraftwerk` placement for `ibm18` was not available. `Kraftwerk` was not run by the author, and run-times were not available.

Circuit	Kraftwerk		FDP	HPWL Ratio
	Pre-Legal HPWL	Overlap (%)	Pre-Legal HPWL	FDP/Kraftwerk
ibm01	1.86	28.65	1.75	0.94
ibm02	4.51	30.28	4.04	0.90
ibm03	5.91	33.19	5.70	0.96
ibm04	7.16	30.37	6.38	0.89
ibm05	12.34	30.19	10.87	0.88
ibm06	5.71	32.29	5.61	0.98
ibm07	10.43	30.73	9.77	0.94
ibm08	10.53	31.16	9.77	0.93
ibm09	12.17	34.69	11.34	0.93
ibm10	19.92	35.13	19.19	0.96
ibm11	17.68	35.82	17.25	0.98
ibm12	26.60	33.54	24.46	0.92
ibm13	22.20	37.85	20.70	0.93
ibm14	38.65	34.40	35.48	0.92
ibm15	50.66	36.12	47.68	0.94
ibm16	52.72	36.80	47.97	0.91
ibm17	78.22	34.24	67.07	0.86
			Average	0.93

4.3.2 Standard Cell Comparisons

The second experiment investigated the stability and capability of FDP to place standard cell designs. To this end, FDP was compared with Capo 8.7 [35] and Dragon [61]. Numerical results from these tests are presented in Table 4.3. FDP was also compared using the same benchmarks but with unit-sized cells, and these results are presented in Table 4.4.

Table 4.3 shows that FDP achieved results that are, on average, 2% better than Capo 8.7, with run times that are within a factor of four for ibm10 and above.[1] It is noted that results are only worse than Capo 8.7 on three of the eighteen benchmark circuits.

In the unit-sized tests, summarized in Table 4.4, FDP proved to be 3% better than Capo 8.7 on average. FDP excels on unit-sized problems for two reasons. First, there is a more even distribution of spreading force magnitudes, so it is less likely that some cells will be pushed "unfairly" to the wrong side of the placement region. Second, the heuristic improvement strategies employed in the legalizer are afforded more alternatives for swapping when cell dimensions are the same size.

4.3.3 Mixed-Size Comparisons

The third experiment investigated FDP's handling of mixed-size designs. Results are compared to those recently published in [81], which include values from Capo, Kraftwerk, and mPG [79]. As the testing platform differed from those used in [81], the relative performance of each approach cannot be compared, although it is possible to compare the quality of the final placements. Performance values are provided as a "reality-check".

This thesis does not explicitly compare to the results of [40]. FDP distributes whitespace throughout the placement area, whereas [40] appears to pack to the left. While the wire lengths of [40] are excellent, the difference in whitespace allocation appears to make comparisons difficult

1 In comparing to Capo, it is important to note that Capo is an actively developed academic placement tool and is always evolving in terms of its version numbers. The version used in this thesis, namely Capo 8.7, is the latest version available at the time of writing. Several other recent papers in the literature have compared to Capo 8.5. Although those results are not presented, the author has compared to Capo 8.5 and has foudn that results are better, on average, by 4% to 5%.

Table 4.3: Results for standard cell circuits with an aspect ratio of 1.0 and 5% whitespace. Cases marked with "n/a" indicates that the tool failed to place the circuit due to a crash. CPU run times are reported in minutes.

Circuit	Capo		Dragon		FDP		vs. Capo	
	WL	CPU	WL	CPU	WL	CPU	WL	CPU
ibm01	1.77	1	1.70	14	1.68	7	0.95	7.0
ibm02	3.87	2	n/a	n/a	3.78	11	0.98	5.5
ibm03	5.38	2	5.73	12	5.56	11	1.03	5.5
ibm04	6.49	3	6.39	25	6.21	15	0.96	5.0
ibm05	10.12	3	9.97	37	10.39	18	1.03	6.0
ibm06	5.59	3	5.71	26	5.38	14	0.96	4.7
ibm07	9.59	5	9.19	35	9.34	23	0.97	4.6
ibm08	9.98	5	9.10	79	9.42	24	0.94	4.8
ibm09	11.51	6	13.03	77	11.23	26	0.98	4.3
ibm10	18.92	8	n/a	n/a	18.36	29	0.97	3.6
ibm11	16.69	8	n/a	n/a	17.04	34	1.02	4.3
ibm12	24.58	9	n/a	n/a	23.24	33	0.95	3.7
ibm13	20.87	11	n/a	n/a	20.51	41	0.98	3.7
ibm14	35.44	18	35.24	151	34.08	67	0.96	3.7
ibm15	46.68	25	49.88	208	46.57	89	1.00	3.6
ibm16	49.02	25	46.52	232	46.38	94	0.95	3.8
ibm17	66.93	30	69.26	505	63.60	120	0.95	4.0
ibm18	45.58	28	46.07	454	45.36	142	1.00	5.1
						Average	0.98	4.6

Table 4.4: Results for unit-sized standard cell circuits with an aspect ratio of 1.0 and 5% whitespace. Cases marked with "n/a" indicates that the tool failed to place the circuit due to a crash. CPU run times are reported in minutes.

Circuit	Capo		Dragon		FDP		vs. Capo	
	WL	CPU	WL	CPU	WL	CPU	WL	CPU
ibm01	2.17	1	2.15	12	2.14	5	0.99	5.0
ibm02	5.22	2	4.97	19	5.21	12	1.00	6.0
ibm03	6.87	2	6.54	21	6.52	9	0.95	4.5
ibm04	7.88	3	7.42	37	7.84	11	0.99	3.7
ibm05	13.04	3	12.39	53	13.78	14	1.06	4.7
ibm06	8.64	3	7.49	45	7.54	16	0.87	5.3
ibm07	12.86	4	11.98	33	12.23	23	0.95	5.8
ibm08	13.64	5	12.92	78	13.37	26	0.99	5.2
ibm09	13.50	6	13.17	63	12.91	22	0.96	3.7
ibm10	22.56	7	21.63	85	21.74	56	0.96	8.0
ibm11	20.46	8	19.33	73	19.82	39	0.97	4.9
ibm12	28.09	8	26.23	93	27.04	67	0.96	8.4
ibm13	24.86	10	24.36	94	23.97	38	0.96	3.8
ibm14	46.32	18	n/a	n/a	43.80	76	0.95	4.2
ibm15	57.37	22	n/a	n/a	57.67	100	1.01	4.6
ibm16	61.82	27	n/a	n/a	58.62	103	0.95	3.8
ibm17	81.53	28	n/a	n/a	77.84	127	0.95	4.5
ibm18	60.43	30	n/a	n/a	58.11	127	0.96	4.2
						Average	0.97	5.0

and potentially misleading.

The results for the macro cell benchmarks are shown in Table 4.5 and Table 4.6. Note that `Capo I` corresponds to the "Improved Flow 1 (C)" of [81], while `Capo II` corresponds to "Flow 2" in [81]. It should be noted that `Capo II` and `Kraftwerk` results are *not* legal whereas the results for `FDP` *are* legalized, and that the ratios comparing `FDP` to `Capo` uses the best value from either of the two `Capo` flows for each circuit.

Compared to the best of the Capo flows, `FDP` achieved results that were 9% better than `Kraftwerk` and 1% better than `mPG`. `FDP`'s run times for these problems also appear to be competitive. These numbers confirm that `FDP` not only implements but extends the techniques presented in [13], rendering placements that are on-par with current state-of-the-art methods.

Table 4.5: Macro cell benchmark results comparing Capo flows to FDP. Run times for Capo are observed on a 2 GHz Pentium. CPU run times for FDP are observed on a 2.8 GHz Pentium, and are reported in minutes.

Circuit	Capo Flow I		Capo Flow II		FDP		WL vs.
	WL	CPU	WL	CPU	WL	CPU	Capo
ibm01	3.36	13	2.92	5	2.62	8	0.90
ibm02	8.23	240	6.5	11	6.59	19	1.01
ibm03	11.53	22	9.63	14	11.17	11	1.16
ibm04	11.93	25	11.2	14	10.05	17	0.90
ibm06	9.63	19	7.9	17	8.89	17	1.13
ibm07	15.80	39	13.6	55	14.09	21	1.04
ibm08	18.85	111	17.2	22	15.93	23	0.93
ibm09	17.52	178	17.8	31	17.89	31	1.02
ibm10	53.58	490	47.5	68	45.44	45	0.96
ibm11	26.47	69	25.1	41	26.60	33	1.06
ibm12	55.12	119	47.5	51	49.72	39	1.05
ibm13	33.56	88	33.4	68	31.83	37	0.95
ibm14	52.67	333	47.9	117	47.72	58	1.00
ibm15	64.69	264	66.8	122	69.00	115	1.07
ibm16	83.14	580	86.7	166	66.82	120	0.80
ibm17	91.50	249	87.6	136	80.35	115	0.92
ibm18	54.11	397	57.2	158	55.56	160	1.03
						Average	1.00

Table 4.6: Macro cell benchmark results comparing `Kraftwerk` and `mPG` to `FDP`. Run times for `Kraftwerk` are observed on a 2 GHz Pentium. Run times for `mPG` are observed on a Sun Blade 1000 running at 750 MHz. Run times for `FDP` are observed on a 2.8 GHz Pentium. All CPU times are reported in minutes.

Circuit	**Kraftwerk**		**mPG**		**FDP**		**WL vs.**	
	WL	**CPU**	**WL**	**CPU**	**WL**	**CPU**	**Kraftwerk**	**mPG**
ibm01	3.01	2	3.01	18	2.62	8	0.87	0.87
ibm02	7.58	9	7.42	32	6.59	19	0.87	0.89
ibm03	11.4	10	11.2	32	11.17	11	0.98	1.00
ibm04	12.1	12	10.5	42	10.05	17	0.83	0.96
ibm06	10.2	12	9.2	45	8.89	17	0.87	0.97
ibm07	17.1	19	13.7	68	14.09	21	0.82	1.03
ibm08	18.2	21	16.4	82	15.93	23	0.88	0.97
ibm09	19.1	28	18.6	84	17.89	31	0.94	0.96
ibm10	51.5	35	43.6	172	45.44	45	0.88	1.04
ibm11	26.6	36	26.5	112	26.60	33	1.00	1.00
ibm12	52.6	43	44.3	153	49.72	39	0.95	1.12
ibm13	35.9	55	37.7	151	31.83	37	0.89	0.84
ibm14	47.4	74	43.5	276	47.72	58	1.01	1.10
ibm15	73.7	93	65.5	285	69.00	115	0.94	1.05
ibm16	82.4	94	72.4	436	66.82	120	0.81	0.92
ibm17	92.2	107	78.5	606	80.35	115	0.87	1.02
ibm18	54.9	110	50.7	437	55.56	160	1.01	1.10
						Average	0.91	0.99

Chapter 5

Conclusion

5.1 Summary and Contributions

This thesis discussed the engineering of an analytic placer inspired by the work of Eisenmann and Johannes [13]. Several implementation details were investigated, including:

- a means of efficiently computing forces for cell spreading;
- algorithms for assessing cell distribution throughout the placement area;
- a method of preventing the destabilization of a placement due to numerical instability; and,
- supplementary techniques for additional wire length minimization.

To the author's knowledge, no other works in the literature have addressed these issues.

This dissertation clarified the intricacies surrounding the implementation of a force-directed heuristic, and introduced new techniques for improving the quality of results. The placements generated by `FDP` were compared to other leading-edge heuristics and found to be favourable, with performance usually within four times that of `Capo 8.7` and results up to 3% better on average. Most importantly, by using this thesis' extensions, `FDP` was shown to produce results up to 9% better than `Kraftwerk`. These results validate the approach and confirm that `FDP` both implements and improves upon the original method.

5.2 Future Directions

There is still considerable room to improve the performance and quality of results from the force-directed heuristic. For example, BoxPlace may be augmented in accordance with [15] so that it does not reintroduce as much overlap late in the placement. This may allow it to further improve the quality of the placement as well as to accelerate cell spreading.

Partitioning one or two levels prior to analytic placement could also improve results. Cutlines added by a partitioning method would not only have the effect of creating a more initially-spread design, they would also render the Hessians more diagonally dominant and therefore faster to solve with a conjugate gradient technique. Lastly, there is still considerable room to improve upon run times through parameter tweaking, and the use of a non-linear stretching technique along the lines of [15].

In addition, the placer and the spread metrics could be extended to consider other placement objectives, such as net congestion and timing. For timing, the Steiner tree method of [59] or the min-max analytic approach of [53] may be incorporated. The Steiner tree method presents the additional benefit that the trees themselves could be used to identify congested regions. The global placer could then adapt forces (or add dummy cells) to discourage these "hot-spots". BoxPlace could also be adapted to account for both wire length- *and* timing-optimal locations for cells. Furthermore, the placer could be extended to consider three-dimensional placement, as in [12]. For this application, it would suffice to replace the quad-tree with an oct-tree and extend the Klee's measure technique to three dimensions, as in [76].

It is hoped that the ideas presented in this thesis will stimulate research in force-directed placement. Most importantly, as this heuristic matures, it is expected that the quality of VLSI CAD tools on a whole will improve in kind.

Bibliography

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

[2] J. Ullman, *Computational Aspects of VLSI*. Computer Science Press, 1984.

[3] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[4] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

[5] D. Perry, *VHDL*. McGraw-Hill, 1998.

[6] K. Shahookar and P. Mazumder, "VLSI cell placement techniques," *ACM Comput. Surv.*, vol. 23, no. 2, pp. 143–220, 1991.

[7] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Field-Programmable Logic and Applications* (W. Luk, P. Y. Cheung, and M. Glesner, eds.), pp. 213–222, Springer-Verlag, Berlin, 1997.

[8] L. McMurchie and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for FPGAs," in *Proc. of the International Symposium on Field Programmable Gate Arrays*, pp. 111–117, ACM Press, 1995.

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

[10] F. Mo, A. Tabbara, and R. K. Brayton, "A timing-driven macro-cell placement algorithm," in *Proc. of the International Conference on Computer-Aided Design*, pp. 322–327, 2001.

[11] C. Mulpuri and S. Hauck, "Runtime and quality tradeoffs in FPGA placement and routing," in *Proc. of the International Symposium on Field Programmable Gate Arrays*, pp. 29–36, ACM Press, 2001.

[12] B. Goplen and S. S. Sapatnekar, "Efficient thermal placement of standard cells in 3D ICs using a force directed approach," in *Proc. of the International Conference on Computer-Aided Design*, pp. 86–89, IEEE Press, 2003.

[13] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. of the Design Automation Conference*, pp. 269–274, ACM Press, 1998.

[14] J. Cong, T. Kong, J. R. Shinnerl, M. Xie, and X. Yuan, "Large-scale circuit placement: Gap and promise," in *Proc. of the International Conference on Computer-Aided Design*, November 2003.

[15] N. Viswanathan and C. C.-N. Chu, "Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," in *Proc. of the International Symposium on Physical Design*, April 2004.

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

[17] W.-J. Sun and C. Sechen, "Efficient and effective placement for very large circuits," *Trans. on Computer-Aided Design*, vol. 14, pp. 349–359, March 1995.

[18] W. Swartz and C. Sechen, "Timing driven placement for large standard cell circuits," in *Proc. of the Design Automation Conference*, pp. 211–215, 1995.

[19] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, pp. 671–680, May 1983.

[20] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and finite-time behavior of simulated annealing," in *Proc. of the Conference on Decision and Control*, pp. 761–767, 1985.

[21] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE Journal of Solid-State Circuits*, pp. 510–522, April 1985.

[22] A. Caldwell, A. B. Kahng, and I. L. Markov, "Optimal partitioners and end-case placers for standard-cell layout," in *Proc. of the International Symposium on Physical Design*, pp. 90–96, ACM Press, 1999.

[23] P. Suaris and G. Kedem, "Standard cell placement by quadrisection," in *Proc. of the International Conference on Computer-Aided Design*, pp. 612–615, 1987.

[24] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: a survey," *Integr. VLSI J.*, vol. 19, no. 1-2, pp. 1–81, 1995.

[25] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Design and implementation of the fiduccia-mattheyses heuristic for VLSI netlist partitioning," in *Proc. of the Workshop on Algorithm Engineering and Experimentation*, January 1999.

[26] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. of the Design Automation Conference*, pp. 175–181, IEEE Press, 1982.

[27] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Sys. Tech. Journal*, vol. 49, no. 2, pp. 291–308, 1970.

[28] A. E. Dunlop and B. W. Kernighan, "A placement procedure for standard-cell VLSI circuits," *Trans. on Computer-Aided Design*, vol. 4, pp. 92–98, January 1985.

[29] G. Karypis, *Multilevel Optimization and VLSICAD*, ch. 3. Boston: Kluwer Academic Publishers, 2002.

[30] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," *Trans. on VLSI*, vol. 7, pp. 69–79, March 1999.

[31] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: application in vlsi domain," in *Proc. of the Design Automation Conference*, pp. 526–529, ACM Press, 1997.

[32] A. E. Caldwell, A. B. Kahng, A. A. Kennings, and I. L. Markov, "Hypergraph partitioning for VLSI CAD: methodology for heuristic development, experimentation and reporting," in *Proc. of the Design Automation Conference*, pp. 349–354, ACM Press, 1999.

[33] J. Vygen, "Four-way partitioning of two-dimensional sets." Unpublished Manuscript, 2000.

[34] J. Vygen, "Algorithms for large-scale flat placement," in *Proc. of the Design Automation Conference*, pp. 746–751, ACM Press, 1997.

[35] A. E. Caldwell, A. B. Kahng, and I. Markov, "Can recursive bisection alone produce routable placements?," in *Proc. of the Design Automation Conference*, pp. 477–482, ACM Press, 2000.

[36] A. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. H. Madden, "Fractional cut: Improved recursive bisection placement," in *Proc. of the International Conference on Computer-Aided Design*, November 2003.

[37] C. J. Alpert, G.-J. Nam, and P. G. Villarrubia, "Free space management for cut-based placement," in *Proc. of the International Conference on Computer-Aided Design*, pp. 746–751, ACM Press, 2002.

[38] S. N. Adya and I. L. Markov, "Improving min-cut placement for VLSI using analytical techniques," in *Proc. IBM ACAS Conference*, pp. 55–62, IBM ARL, February 2003.

[39] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *Trans. on Computer-Aided Design*, vol. 10, pp. 356–365, March 1991.

[40] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh, and P. H. Madden, "Recursive bisection based mixed block placement," in *Proc. of the International Symposium on Physical Design*, April 2004.

[41] A. B. Kahng and S. Reda, "Placement feedback: a concept and method for better min-cut placements," in *Proc. of the Design Automation Conference*, pp. 357–362, ACM Press, 2004.

[42] R.-S. Tsay, E. Kuh, and C.-P. Hsu, "PROUD: A sea-of-gates placement algorithm," *IEEE Design and Test of Computers*, vol. 5, December 1988.

[43] H. Etawil, S. Areibi, and A. Vannelli, "Attractor-repeller approach for global placement," in *Proc. of the International Conference on Computer-Aided Design*, pp. 20–24, 1999.

[44] G. Sigl, K. Doll, and F. Johannes, "Analytical placement: A linear or a quadratic objective function?," in *Proc. of the Design Automation Conference*, pp. 427–432, June 1991.

[45] C. Alpert, T. F. Chan, D. J. Huang, A. B. Kahng, I. Markov, P. Mulet, and K. Yan, "Faster minimization of linear wirelength for global placement," in *Proc. of the International Symposium on Physical Design*, pp. 4–11, April 1997.

[46] J. Li, J. Lillis, L.-T. Liu, and C.-K. Cheng, "New spectral linear placement and clustering approach," in *Proc. of the Design Automation Conference*, pp. 88–93, ACM Press, 1996.

[47] A. Kennings and I. Markov, "Analytical minimization of half-perimeter wirelength," in *Proc. of the Asia and South Pacific Design Automation Conference*, pp. 179–184, ACM/IEEE, January 2000.

[48] R. Baldick, A. B. Kahng, A. Kennings, and I. Markov, "Function smoothing with applications to VLSI layout," in *Proc. of the Asia and South Pacific Design Automation Conference*, January 1999.

[49] A. Kennings and I. Markov, "Analytical placement of hypergraphs," Tech. Rep. 990020, UCSD VLSI CAD Laboratory, January 1999.

[50] K. M. Hall, "An *r*-dimensional quadratic placement algorithm," *Management Science*, vol. 17, pp. 219–229, November 1970.

[51] P. M. Hahn, W. L. Hightower, T. A. Johnson, M. Guignard-Spielberg, and C. Roucairol, "Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem," *Yugoslav Journal of Operations Research*, vol. 11, no. 1, pp. 41–60, 2001.

[52] C.-K. Cheng and E. Kuh, "Module placement based on resistive network optimization," *Trans. on Computer-Aided Design*, vol. 3, pp. 218–225, July 1984.

[53] A. B. Kahng, S. Mantik, and I. L. Markov, "Min-max placement for large-scale timing optimization," in *Proc. of the International Symposium on Physical Design*, pp. 143–148, April 2002.

[54] S. T. Obenaus and T. H. Szymanski, "Gravity: Fast placement for 3-D VLSI," *Trans. on Design Automation of Electronic Systems*, vol. 8, no. 3, pp. 298–315, 2003.

[55] A. B. Kahng and Q. Wang, "Implementation and extensibility of an analytic placer," in *Proc. of the International Symposium on Physical Design*, pp. 18–25, April 2004.

[56] W. Naylor, R. Donelly, and L. Sha, "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.

[57] S. S. Sapatnekar and S.-M. Kang, *Design Automation for Timing-Driven Layout Synthesis*. Kluwer Academic Publishers, 1992.

[58] Z. Xiu, J. D. Ma, S. M. Folwer, and R. A. Rutenbar, "Large-scale placement by grid-warping," in *Proc. of the Design Automation Conference*, pp. 351–356, ACM Press, 2004.

[59] B. Obermeier and F. M. Johannes, "Quadratic placement using an improved timing model," in *Proc. of the Design Automation Conference*, pp. 705–710, ACM Press, 2004.

[60] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM Journal on Applied Mathematics*, vol. 14, pp. 255–265, 1966.

[61] X. Y. M. Wang and M. Sarrafzadeh, "Dragon2000: Standard-cell placement tool for large industry circuits," in *Proc. of the International Conference on Computer-Aided Design*, November 2000.

[62] F. Mo, A. Tabbara, and R. K. Brayton, "A force-directed macro-cell placer," in *Proc. of the International Conference on Computer-Aided Design*, pp. 177–180, EECS, UC Berkeley, November 2000.

[63] B. Hu and M. Marek-Sadowska, "FAR: Fixed-points addition & relaxation based placement," in *Proc. of the International Symposium on Physical Design*, pp. 161–166, ACM Press, 2002.

[64] S.-W. Hur, T. Cao, K. Rajagopal, Y. Parasuram, A. Chowdhary, V. Tiourin, and B. Halpin, "Force directed Mongrel with physical net constraints," in *Proc. of the Design Automation Conference*, pp. 214–219, ACM Press, 2003.

[65] A. Caldwell, A. B. Kahng, and I. L. Markov, "Toward CAD-IP reuse: The MARCO GSRC bookshelf of fundamental CAD algorithms," *IEEE Design and Test of Computers*, vol. 19, no. 3, pp. 70–79, 2002.

[66] J. G. Siek and A. Lumsdaine, "The matrix template library: Generic components for high-performance scientific computing," *Computing in Science and Engineering*, vol. 1, pp. 70–78, November/December 1999.

[67] L.-Q. Lee, *Generic Programming for High-Performance Scientific Computing*. Ph. D. thesis, University of Notre Dame, 2002.

[68] "The Boost C++ library." `http://www.boost.org`, Current July 2004.

[69] U. T. Mello and I. Khabibrakhmanov, "On the reusability and numeric efficiency of C++ packages in scientific computing," in *Proc. of the ClusterWorld Conference and Expo*, June 2003.

[70] C. J. Alpert, T. Chan, D. J.-H. Huang, I. Markov, and K. Yan, "Quadratic placement revisited," in *Proc. of the Design Automation Conference*, pp. 752–757, 1997.

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

[72] Y. Saad, "ILUT: A dual threshold incomplete ILU factorization," Tech. Rep. 92-38, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 1992.

[73] J. George, *Computer Implementation of the Finite-Element Method*. Ph. D. thesis, Stanford University, 1971.

[74] L. Hagen and A. Kahng, "Improving the quadratic objective function in module placement," in *Proc. of the IEEE International ASIC Conference and Exhibit*, pp. 21–25, September 1992.

[75] J. Barnes and P. Hut, "A hierarchical $O(n \log n)$ force calculation algorithm," *Nature*, vol. 324, 1986.

[76] M. H. Overmars and C.-K. Yap, "New upper bounds in Klee's measure problem," *SIAM Journal on Computing*, vol. 20, no. 6, pp. 1034–1045, 1991.

[77] J. Bentley, "Algorithms for Klee's rectangle problems." Unpublished Manuscript, 1977.

[78] T. F. Chan, J. Cong, T. Kong, J. R. Shinnerl, and K. Sze, "An enhanced multilevel algorithm for circuit placement," in *Proc. of the International Conference on Computer-Aided Design*, November 2003.

[79] C.-C. Chang, J. Cong, and X. Yuan, "Multi-level placement for large-scale IC designs," in *Proc. of the Asia and South Pacific Design Automation Conference*, pp. 325–330, 2003.

[80] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *Trans. on VLSI*, vol. 11, pp. 1120–1135, December 2003.

[81] S. Adya and I. Markov, "Combinatorial techniques for mixed-size placement," *Trans. on Design Automation of Electronic Systems*, 2004. To appear.

[82] S. Adya and I. Markov, "Consistent placement of macro-blocks using floorplanning and standard-cell placement," in *Proc. of the International Symposium on Physical Design*, ACM Press, 2002.

[83] A. Saurabh and I. Markov, "ISPD02 mixed-size placement benchmarks." `http://vlsicad.eecs.umich.edu/BK/ISPD02bench`, Current July 2004.

Glossary

BiCGStab	*Bi-Conjugate Gradient-Stable Algorithm*.
BookShelf	An Internet-based repository for literature, benchmarks, and files related to VLSI CAD. Available at: `http://www.gigascale.org/bookshelf`.
Boost	A set of free, peer-reviewed, portable C++ source libraries, available at: `http://www.boost.org`.
Capo	A recursive, minimum-cut bi-partitioning placement tool, available at: `http://vlsicad.eecs.umich.edu/BK/PDtools`.
CAD	*Computer Aided Design.*
CPU	*Central Processing Unit*. Software runtimes, 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`.

FDP	*Force Directed Placer*. The force-directed placement tool developed as part of this thesis.
FM	*Fidduccia-Mattheyses*. A bi-partitioning heuristic described in [1].
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 lookup 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*.
ILU(0)	*Incomplete LU*. An LU matrix preconditioner without threshold or fill-in, described in [2].
I/O	*Input/Output*.
KL	*Kernighan-Lin*. A bi-partitioning heuristic described in [3].
Kraftwerk	A commercial force-directed placer based on [4].

Macro Cell	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 macro cells.
mPG	A multilevel placement tool described in [5].
MTL	*Matrix Template Library*. A C++-based library for matrix computation, available at: `http://www.osl.iu.edu/research/mtl`.
NP	*Non-deterministic Polynomial*. A set of computational decision problems which are solvable by a nondeterministic 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 [6, 7].
Reverse Cuthill-Mckee	A matrix reordering heuristic described in [8].
Standard Cell	A cell whose dimensions are specified in a standard library.
VLSI	*Very Large Scale Integration*.

uBLAS	A C++-based matrix library. Part of the Boost C++ distribution.
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] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. of the Design Automation Conference*, pp. 175–181, IEEE Press, 1982.

[2] Y. Saad, "ILUT: A dual threshold incomplete ILU factorization," Tech. Rep. 92-38, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 1992.

[3] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Sys. Tech. Journal*, vol. 49, no. 2, pp. 291–308, 1970.

[4] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. of the Design Automation Conference*, pp. 269–274, ACM Press, 1998.

[5] T. F. Chan, J. Cong, T. Kong, J. R. Shinnerl, and K. Sze, "An enhanced multilevel algorithm for circuit placement," in *Proc. of the International Conference on Computer-Aided Design*, November 2003.

[6] P. M. Hahn, W. L. Hightower, T. A. Johnson, M. Guignard-Spielberg, and C. Roucairol, "Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem," *Yugoslav Journal of Operations Research*, vol. 11, no. 1, pp. 41–60, 2001.

[7] K. M. Hall, "An *r*-dimensional quadratic placement algorithm," *Management Science*, vol. 17, pp. 219–229, November 1970.

[8] J. George, *Computer Implementation of the Finite-Element Method*. Ph. D. thesis, Stanford University, 1971.