A Clustering Utility-Based Approach for ASIC Design

by

Matt D. Thompson

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical & Computer Engineering

Waterloo, Ontario, Canada, 2001

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

This thesis presents two new approaches for dealing with the high complexity of ASIC design. A novel utility-based search technique is applied to iterative improvement in the standard-cell placement problem. Utility theory is used to guide a deterministic (greedy) search heuristic in finding a local minimum quickly by ranking moves based on an estimate of their proximity to an optimal location. Moves are then chosen that are statistically more likely to improve than if the moves were chosen randomly, greatly increasing the rate of convergence. Then a new hierarchal clustering heuristic is presented which clusters a standard-cell circuit by greedily collapsing net hyperedges by size, but not permitting very large clusters from forming. The clustering heuristic demonstrates excellent characteristics for reducing the execution time of standard-cell placement while achieving better results compared to non-clustered circuit placement and placement using other edge-based clustering methods.

Acknowledgements

First and foremost, I would like to thank my family. To my fiance, Miki, thank you for your love and support throughout this thesis. This effort truly would not have been possible if it were not for you. And to my parents, thank you for encouraging me and making me feel worthy of your pride.

I would like to thank Professor Shawki Areibi for his encouragement and vision. He made me strive for excellence at every point of this work. His comments and requirements vastly improved this research.

I would like to thank Professor Anthony Vannelli for his guidance and insight. His was a voice of calm reason at every critical junction, and kept me from losing sight of what it was that I was trying to accomplish.

I would also like to thank Andrew Kennings for encouraging me to attempt a Master's degree in the first place, even after he had moved on to greener pastures.

And finally, thank you to my good friends and colleagues Bill, Chris, Mattias, Moataz, Andrew, Hussein, Laleh, and Min, who each contributed in their own way..

Contents

1 Introduction **1**

1.1 The VLSI Design Problem 1

1.2 The VLSI Design Process 2

1.2.1 Behavioral Design 3

1.2.2 Functional Design 4

1.2.3 Logical Design 4

1.2.4 Circuit Design 4

1.2.5 Physical Design 4

1.2.6 Fabrication and Testing 4

1.3 Design Goals 5

1.3.1 Area 5

1.3.2 Speed 5

1.3.3 Power Dissipation 6

1.3.4 Design Time 6

1.3.5 Testability 6

1.4 Motivation and Contributions 6

1.5 Thesis Organization 8

2 Background **10**

2.1 Introduction 10

2.2 Physical Design 11
2.2.1 Physical Design Process 11
2.2.2 Layout Styles 12
2.3 Standard-Cell Placement 17
2.3.1 Problem Overview 17
2.3.2 Interconnection Cost 18
2.3.3 Placement Quality Measure 19
2.3.4 Placement Problem and the Traditional Quadratic Measure 20
2.4 Approaches for Solving the Standard-Cell Problem 21
2.4.1 Numerical Optimization 22
2.4.2 Min-Cut Placement 23
2.4.3 Constructive Placement 24
2.4.4 Iterative Improvement (Search Heuristics) 24
2.5 Test Circuits 26
2.6 Summary 28

3 Utility Function-Based Iterative Search 30

3.1 Introduction 30
3.2 Previous Work 31
3.3 Solution Methodology 33
3.3.1 Utility of Net Placement 33
3.3.2 Wirelength Lower Bound 34
3.3.3 Cell Utility 36
3.3.4 The UTILITY Heuristic 38
3.3.5 Move Selection 39
3.3.6 Local Search Window 41
3.3.7 Gain Evaluation 42
3.4 Results 46
3.4.1 UTILITY as a Primary Improver 46

3.4.2 UTILITY as a Post-Processing Improver 48
3.4.3 Local Search Window Size 50
3.4.4 Cell Utility Net Size Cut-off 50
3.4.5 Effectiveness of Utility as a Measure of Cell Placement Quality 52
3.5 Summary 54

4 Clustering-based Placement **57**

4.1 Introduction 57
4.1.1 Motivation 57
4.1.2 Circuit Coarsening 58
4.1.3 Clustering v. Coarsening 59
4.1.4 Measurement of clustering quality 59
4.1.5 Goals of Clustering and De-clustering 60
4.1.6 Contributions 61
4.2 Previous Work 61
4.2.1 Partitioning-based Clustering methods 62
4.2.2 Constructive Clustering methods 64
4.2.3 Hierarchal Clustering 66
4.2.4 Application to the Standard-Cell Placement Problem 67
4.3 Clustering-based Standard-Cell Placement 68
4.3.1 Weighted Hyperedge Clustering 68
4.3.2 Implementation as a mapping function 70
4.3.3 Reduction of Nets 72
4.3.4 Pin Placement 74
4.3.5 De-clustering 75
4.3.6 Top-Level Improvement 78
4.4 Results 79
4.4.1 Test Circuits 79
4.4.2 WHEC Clustering Performance 80

4.4.3 Clustering Method Comparison 81
4.4.4 Effect of Cell Statistics 84
4.4.5 Clustering Depth 86
4.4.6 FLATTEN De-clustering Heuristic 87
4.4.7 UTILITY De-clustering Improvement 90
4.5 Summary 90

5 Conclusions **92**

Bibliography **96**

List of Tables

2.1 Benchmarks used for testing 27

2.2 Connectivity and Cell Sizes of Benchmarks 27

3.1 UTILITY performance 50

4.1 Wirelength and Run-Time Comparison, FLAT Versus WHEC 81

4.2 Wirelength Comparison, WHEC Versus EC and MHEC 82

4.3 Run Time Comparison, WHEC Versus EC and MHEC 83

4.4 Cell and Net reduction comparison 83

4.5 Size and Connectivity Statistics, EC Clustering 84

4.6 Size and Connectivity Statistics, MHEC Clustering 85

4.7 Size and Connectivity Statistics, WHEC Clustering 85

4.8 FLATTEN de-clustering heuristic performance 89

4.9 UTILITY improver de-clustering effectiveness 90

List of Figures

1.1 Common VLSI Design Process Steps 3

1.2 Multi-level Standard-Cell Placement Task 9

2.1 Common VLSI Layout Styles 13

2.2 Half-perimeter Wirelength is an approximation of the routing cost of a net 18

2.3 Taxonomy of Solution Approaches to Layout Problems 22

3.1 wirelength Bound Computation 35

3.2 Large nets are difficult to Improve 37

3.3 Net Size Distribution 39

3.4 Utility-based Iterative Improver Heuristic 40

3.5 Determining cell shift direction 44

3.6 Estimating change in HPWL for small cell displacements 45

3.7 Comparison of UTILITY to TILE as a primary improver 47

3.8 Effect of search depth 49

3.9 Effect of window size 51

3.10 Effect of Net Size Cut-off in Cell Utility Calculation 52

3.11 Correlation between improvement and cell utility 53

3.12 Distribution of total half-perimeter wirelength by net size 54

3.13 Average net utility by net size. 55

4.1 Multilevel Clustering Hierarchy 66

4.2 Weighted Hyperedge Clustering 70
4.3 Mapping Implementation 71
4.4 Reduction of Internal Nets 73
4.5 Net Reduction Algorithm 74
4.6 Pin Placement in a Cluster 75
4.7 Ordering Cells by Sorting Pin Positions 77
4.8 Wirelength Versus Clustering Depth, struct Benchmark 86
4.9 Wirelength Versus Clustering Depth, bio Benchmark 87
4.10 Wirelength Versus Clustering Depth, avq.large Benchmark 88
4.11 Clustering method effects on circuit size reduction, avq.large Benchmark 88

Chapter 1

Introduction

1.1 The VLSI Design Problem

Modern electronics have reached a point of unparalleled complexity. Microprocessors in a typical home computer today contain tens of millions of transistors. When that microprocessor is being designed, each one of its transistors must be designed and physically laid out on a microchip in order to implement the circuitry. The only reason that this complexity can be handled at all is because modern design tools are used that significantly reduce the burden on the design engineer.

The phrase associated with the task of automatically designing a circuit using software tools is Design Automation (DA). The ultimate goal of the DA research field is to fully automate the tasks of designing, verifying, and testing a circuit. Some day in the future, perhaps, a design engineer will be able to describe the desired functionality to a software tool, and the tool will provide all the information needed to manufacture a microchip satisfying all requirements.

Unfortunately, we are still a long ways from this goal. No software package is currently capable of handling the numerous, often contradicting, design goals required of modern ICs. To complicate matters further, many different target technologies exist to fabricate a chip and each can involve very different design considerations. For example, in a full-custom design (layout styles are described later in section 2.2.2), minimizing layout area size is of the foremost consideration

in a layout, while in an array-type design, the layout area size is dictated by the pre-fabricated chip used, and so is of no consequence to the designer. This type of dichotomy surfaces in many aspects of DA.

1.2 The VLSI Design Process

The design problem is too complicated for a software tool to attack at once. Complexity theory would seem to suggest that this will always be the case. So we are left with a dilemma. The only feasible approach to solving the VLSI design problem is a divide-and-conquer strategy in which the design task is broken down into independent sub-tasks that are in themselves more tractable to a software tool. Decisions in the design flow can then be affected by a design engineer who's experience can guide a design to a (hopefully) acceptable solution. Of course, using a divide-and-conquer approach does not guarantee an optimal, or even a "good" solution. In practice, it results in layouts that are "good enough", according to the design goals being observed.

There are a series of steps that are often used to design a VLSI circuit. An outline of the most common steps [SaiYou95] is illustrated in Figure 1.1. Note that Figure 1.1 shows VLSI design as a linear process, with one step leading to the next until finally a chip is produced. In reality, design is a highly iterative process, as often one or more steps must be repeated to meet the requirements of a previous design step.

There are many reasons for separating the design process into steps. Firstly, by separating the process into independent steps, the complexity of the problem is reduced. For example, once the functional description of a circuit has been determined, the logic description is formulated based on the functional units, not directly from the requirements. Secondly, the specifications imposed by a higher level onto a lower level can more easily be used to verify the circuit. For example, when designing an adder, we don't need to consider the circuit requirements, only the functional description of the adder. Therefore, by abstracting the previous step to the next step, the designer can focus his or her efforts on a smaller, more manageable task. The steps of Figure 1.1 are discussed separately in the following sections.

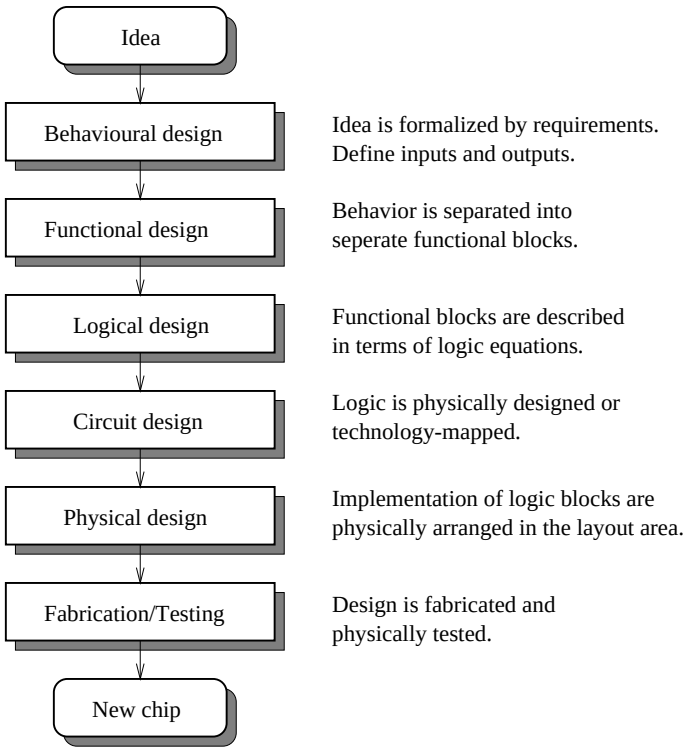

Figure 1.1: Common VLSI Design Process Steps

1.2.1 Behavioral Design

The process of designing a VLSI circuit begins with an idea for a new chip. The first step of the design process to implement that idea is the behavioral design step, in which the idea is translated into a formal description of what the new design should do. The items in this description are the circuit's *requirements*. This description does not necessarily include all behavior of the final manufactured microchip, but rather the subset of behavior that is necessary and sufficient to implement the idea. Later design steps "fill in" any unspecified behavior in whatever way turns out to be easiest to design, provided that the original requirements are not contradicted. Because the description at this design step is purely behavioral, rather than structural, it can be completely independent of the technology that will eventually implement it.

1.2.2 Functional Design

After the requirements are specified, the functional description of the circuit is determined. This is still a high-level description, where the circuit is divided into functional units that each perform a specific task or sets of tasks. This level is still technology independent, but some decisions made at this level will affect the final circuit layout.

1.2.3 Logical Design

After the functional description is complete, a logic description is made to implement each functional unit. At this stage of the design, the logical behavior of the circuit is described in terms of primitive logic operations (NAND, NOT, etc.).

1.2.4 Circuit Design

The circuit design level creates a technology-dependent description of the logic in a circuit. At this level, electrical equations dictate the behavior of the circuit. All logic gates and memory units are represented by transistors and passive elements. In some implementation topologies, logic equations are broken down and mapped to available physical circuit blocks in the circuit topology (called *technology mapping*), or pre-designed logic circuit implementations (e.g., a standard-cell library). In other topologies, individual gates must be designed to implement the logic.

1.2.5 Physical Design

The final step of the design is the physical step. In this step, the circuit-level implementation of logic blocks, as described in the circuit design step, are physically arranged on a chip surface. The end result of the physical design stage is a representation of how to actually fabricate the design.

1.2.6 Fabrication and Testing

After the design is complete, the circuit must go through the fabrication and testing stages. The fabrication stage is where the circuit is actually manufactured and packaged. The testing stage

verifies that the chip meets the specifications laid out previously in the requirements stage. It is important to recognize that, although the last step is named the "testing" stage, we are referring here to testing that the physical circuit meets expected behaviour. Other steps also involve testing and verification processes to verify that the design steps' results meet the previous design steps' requirements. If any design step fails to meet its requirements, either the design process for that step repeats (iterates), or the design process fails.

1.3 Design Goals

There are numerous design goals in VLSI design automation. These goals guide us in our search for good heuristics, and give us a measure of the quality of a circuit layout. What follows are brief descriptions of the most common design goals [Ger99].

1.3.1 Area

Minimizing the area of a design reduces the actual material used, and so more chips can be produced from a single wafer. In itself, this is of little value, but a corollary of this property is that the yield of the wafer, that is, the proportion of working chips on the wafer, increases. Since defects usually occur as a function of area and are fairly randomly distributed, a smaller chip area means that more chips will be free of defects.

1.3.2 Speed

The faster the chip performs, the more attractive it is to industry. Certain functions might also be dependent on the circuit's minimum speed performance, in which case the speed can be a constraint rather than a design goal to be maximized. Speed is often very difficult to measure during design, and is highly technology dependent, so approximations are usually used.

1.3.3 Power Dissipation

Minimizing power dissipation is a goal of increasing importance as portable electronics become more heavily relied upon, since reduced power consumption can mean a longer battery life for laptops and cell-phones. Also, localized regions of high power dissipation, called hot-spots, can lead to early chip failure. Similar to speed, power dissipation can be very difficult to use as a design goal to be minimized, and so is often used as a design constraint.

1.3.4 Design Time

One factor easily overlooked is the time for design. Particularly for small production runs, the engineering costs for actually designing a circuit can be considerable. In other cases, a fast prototype may be desired to test an idea in hardware. A faster tool usually means poorer layouts, but in these situations, a fast design can be more valuable than an optimal one. The use of good automation tools can reduce the design time.

1.3.5 Testability

In any fabrication process, some chips will be defective. In complex chips, these defects may not be readily apparent, but they still exist. Having an easily testable chip can save considerable expense, since test equipment is very expensive and time consuming to use. Making a chip more testable usually means an increase in area, lowering chip yield and increasing cost.

1.4 Motivation and Contributions

The design automation task is enormously complex. Even after three decades of research, typical design tools measure their run-time in terms of hours or even days (e.g., [SunSec95]).

Many previous tools used mathematically demanding techniques to get high quality solutions, taking advantage of the predominantly small problem size being used industrially at the time. In more recent years, the problem sizes used in industry have grown by orders of magnitude.

For very large problems, computationally expensive techniques cannot function in any reasonable amount of time.

However, as we move to deep sub-micron designs below 0.18 microns, the delay of a circuit, as well as power dissipation and area, is dominated by interconnections between logical elements (i.e. transistors)[BelElm95]. Since the delay (and therefore area) of a circuit cannot be ignored, work must still be done to reduce the area of placement and routing for high performance designs. Poor solutions, even if arrived at quickly, are useless in industry. Therefore, there is a great need for design automation tools that operate in a reasonable amount of time, while still arriving at "reasonably good" solutions.

The work in this thesis presents two new methods for handling the highly complex task of designing ASIC chips. The methods are general in nature, and are applicable to many different circuit layout topologies. The usefulness of these methods is demonstrated by applying them to the design task of cell placement for standard-cell topologies. The goal, therefore, is to use these techniques to reduce the execution time of the placement task while providing similar-quality solutions to traditional placement techniques. Figure 1.2 shows the context of these contributions, in relation to design automation in general and the physical design task in specific. This work primarily targets the area and design time goals of section 1.3.

First, we take advantage of recent developments in utility theory to develop a novel utility-based iterative search heuristic. This method defines a metric, *cell utility*, to measure the quality of a cell's placement. By focusing on reducing cell utility, the placement quality of poorly placed cells is improved, exploring the search space of the problem more efficiently. Using utility permits tremendous time savings compared to traditional methods using randomized search methods.

Second, we adapt multilevel clustering techniques from the circuit partitioning task to the standard-cell placement task. Very little work has previously been done in the area of clustering-based placement, despite its high performance record when applied to the partitioning problem. Clustering-based partitioning in now the *de-facto* standard for obtaining high-quality solutions, and is widely accepted as the only practical means to tackle large design problems [AlpHua98]. The new method of clustering a circuit presented here reduces the placement problem size, creating

a much smaller search space that can be investigated more rigorously and in less time than the original (flat) circuit can. The method is based on clustering nets (hyper-edges) in the original circuit, while introducing a new technique for obtaining balanced cluster sizes.

1.5 Thesis Organization

The next section formalizes the context and scope of the standard-cell placement problem. The sub-tasks making up the physical design task of VLSI design are introduced, and the layout technologies that affect physical design are described. Then, the standard-cell placement problem is formally defined, and standard approaches to solve the problem are briefly surveyed.

In chapter 3, a utility-based iterative search heuristic is presented, and applied to improving a standard-cell placement. In Chapter 4, several multi-level clustering techniques are described, and a new fast clustering method is described. Numerical results are presented for both techniques. Finally, in chapter 5, conclusions are drawn, and future directions for this research are discussed.

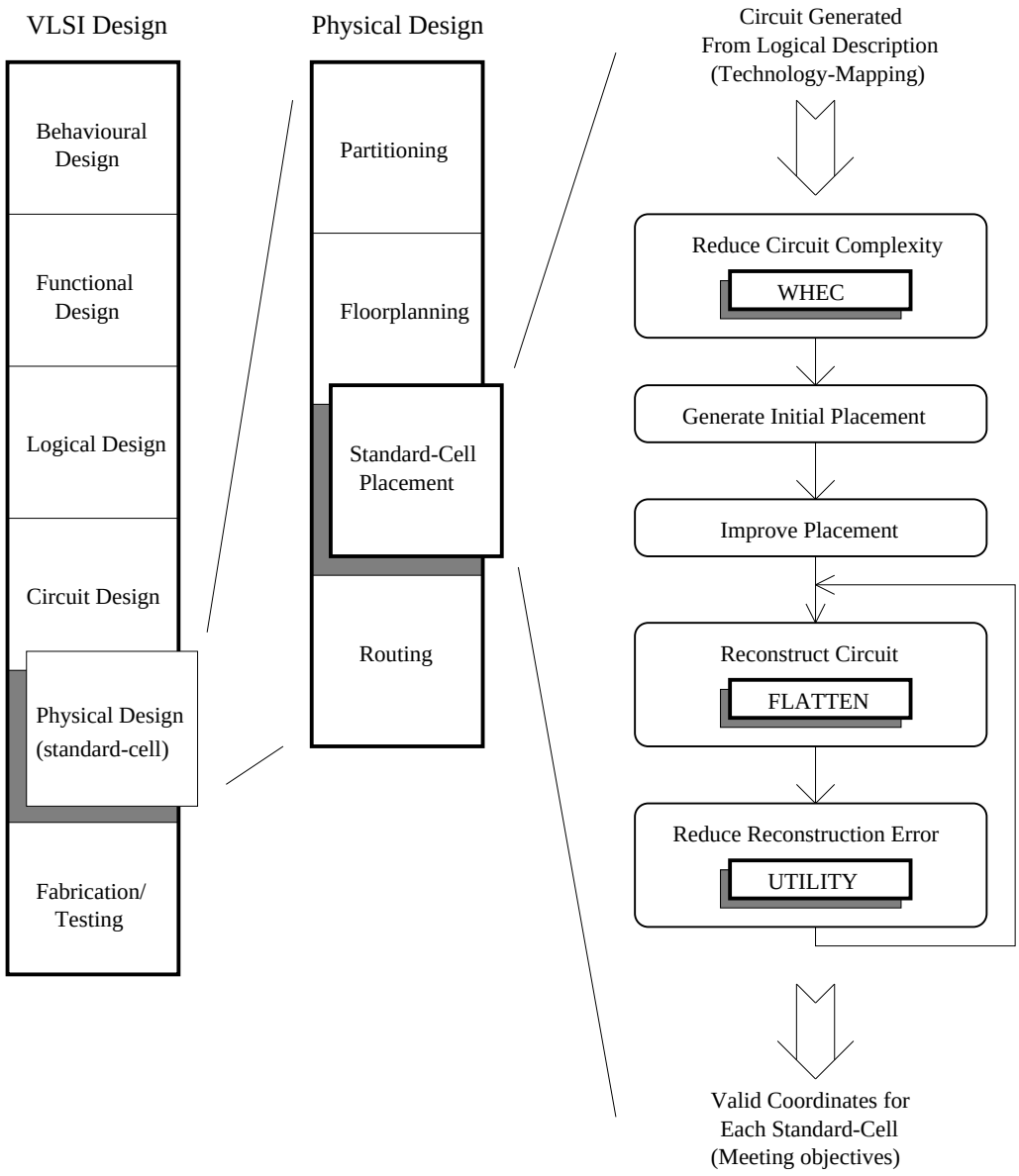

Figure 1.2: Multi-level placement task for standard-cell placement, showing location within VLSI design process and highlighting thesis contributions (WHEC weighted hyper-edge clustering heuristic, UTILITY utility-based iterative improver, and FLATTEN greedy de-clustering heuristic)

Chapter 2

Background

2.1 Introduction

The VLSI design process is broken down into simpler, more computationally tractable tasks, to manage the high complexity of the task. One of these tasks is physical design, which, while a much simpler task when handled independently, is still incredibly complex. Not surprisingly, this complexity is handled by dividing the physical design task into more tractable sub-tasks.

One sub-task within physical design is placement, in which technology-mapped logic components are arranged on a chip. This is a particularly demanding task for automation, especially for the standard-cell layout style. In this style there are many physical parameters of the layout that are unconstrained, making high-quality layout automation very difficult and time-consuming.

This thesis presents search and complexity-management techniques that are applied to the standard-cell layout style. In the remainder of this chapter, the context and scope of the standard-cell placement problem is presented. Section 2.2 describes the physical design task, and describes the division of tasks made by automated layout tools. The layout styles that dictate the heuristics that are then presented in section 2.2.2. Section 2.3 describes the standard-cell placement problem and gives a mathematical formulation for it. Section 2.4 describes some approaches that have been taken in the past to solve the standard-cell placement problem. Finally, section 2.5 describes the

benchmark circuits that are used throughout this thesis to measure and compare performance of the work presented.

2.2 Physical Design

Like the VLSI design process, physical design automation is divided into more tractable sub-tasks, so that a physical design can be realized in a reasonable amount of time. The sub-tasks that are used depend on many factors such as the target technology, the layout style used, the desired design time, the desired chip size, and so on. For example, a small circuit usually doesn't need to be partitioned to fit onto several chips, since it can probably fit on a single chip.

2.2.1 Physical Design Process

The physical design process usually performs most or all of the following steps: partitioning, placement, and routing.

Partitioning

If an initial design is too large to be placed in a single target chip, the problem must be partitioned into several sub-circuits. The most common metric of the quality of a partitioning solution is the number of interconnections between partitions (e.g., [FidMat82]).

Partitioning has been a very active topic for research in physical design automation, and many algorithmic techniques for other stages of physical design and general design automation originated in applications to partitioning (for a recent survey on partitioning, see [AlpKah95]).

Placement

After the circuit is assigned to a placement area of sufficient size, the layout modules of the circuit must be physically assigned, or "placed", in valid positions in the placement area. The placement task is highly dependent on the layout style used.

Routing

In the routing step, interconnections between layout modules are physically assigned to allowable routing "tracks". Routing connections directly to physical tracks has been attempted with limited success, but the more common approach is to further divide the routing problem into global routing and detailed routing. Global routing assigns interconnections a general path through a chip. In this stage, the primary goal is to minimize congestion through any one part of the chip. The second step, detailed routing, attempts to find a valid track assignment, using the results from the global routing stage to simplify the search.

2.2.2 Layout Styles

A VLSI design includes logical and physical designs of a circuit. The logical design of a circuit is independent of an implementation, while the physical design is inherently linked to the layout style of the target technology that will implement the desired behavior. The layout style dictates many design constraints for physical design, since it must be possible to fabricate the physical design in the desired technology. Different design styles are used to alter the design style to achieve some quality gain. For example, highly-constrained design styles are used to simplify and speedup the design process, while loosely-constrained design styles are used to achieve high-performance, low-area designs.

There are currently many different technologies that can implement a VLSI design. Figure 2.1 illustrates several of the most common layout styles currently in use.

Custom Layout

The most basic, and also one of the most used, physical design method is for a design engineer to lay out an entire circuit by hand. The chip topology for this style of design is called *full-custom layout*. An example of full-custom design is shown in Figure 2.1(D). Since this layout is completely at the control of the designer, very compact and efficient layouts can be obtained. Usually the only limitations imposed on the designer in full-custom layout are the physical design rules of the fabrication technology used.

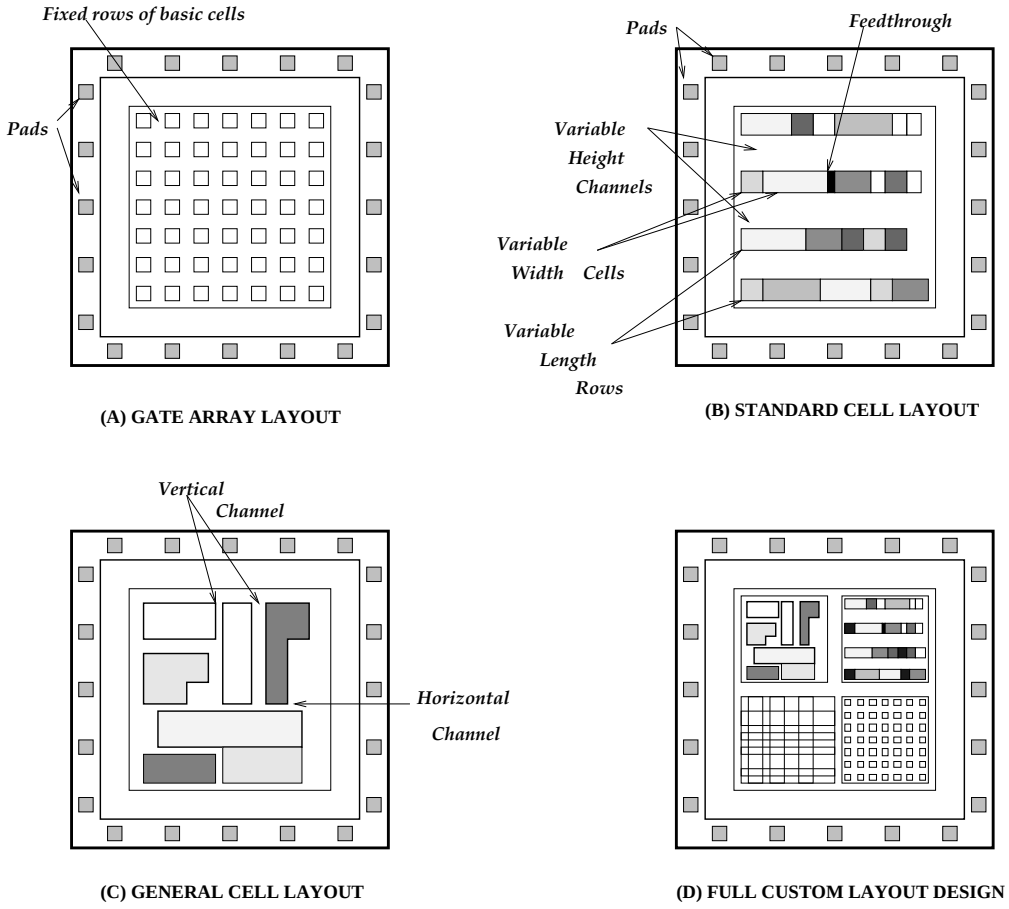

Figure 2.1: Common VLSI Layout Styles

This topology permits the greatest flexibility of design and results in the smallest chip area. It is also the most complicated to layout, and therefore the most time-consuming (and expensive). It is also the slowest to fabricate. Because of the prohibitive design costs involved, the full-custom layout style is only suitable for large production run chips and for relatively small chips (in the SSI or MSI domain) or for small sub-components within a chip (in the VLSI domain).

However, this flexibility allows another approach for full-custom design. Normally a full-custom design is laid-out as blocks of other layout styles, where each block is matched to the layout style which best represents it. For example, a logic array might be laid-out as a gate-array (discussed in the next section), while a memory array would most likely be laid-out by hand for

speed and space efficiency. Full-custom layout is often called *mixed layout* when it approached in this manner.

This can be a very powerful design style, and is the style used in industry for very large, very high production chip design (such as a personal computer microprocessor). It is also very difficult to automate full custom design. Typically, the individual blocks will be floor-planned by making early assumptions about their connections to other blocks. These assumptions become design constraints when designing the internal circuitry of the blocks. Several iterations might have to be made before a legal solution is obtained.

Because of the high engineering costs involved in designing in the full-custom style, it is not well-suited for small production runs. It is also not suited for situations requiring fast fabrication turn-around, such as just-in-time production or fast prototyping, since the entire chip must be fabricated (in comparison to gate-array styles where most or all of the chip is pre-fabricated).

Gate-Array

Gate-array layout is actually a term given to a set of topologies, such as sea-of-gates (SOG), mask-able gate-array (MGA) and a number of other gate-array topologies. Gate-array layouts are highly structured topologies, generally consisting of a grid array of pre-fabricated generalized logic blocks, as shown in Figure 2.1(A). By programming or fabricating interconnections, the desired design can be implemented. For example, in a mask-able gate-array layout, interconnections (wires) and logic-connecting metal layers are laid out as a set of photo-lithography masks, which are used to fabricate the final chip.

Gate-array layouts are well suited for design automation, due to their regular layout style. Chip sizes and valid placement locations are fixed by the type of gate-array used, and cannot be altered by the automation software. Also, because most or all of the physical chip is pre-fabricated, the turn-around time is much faster than that of full-custom layout, making gate-arrays suitable for medium-sized production. However, the mask-able versions of the gate-array topology must still be sent out to fabrication, making them unsuitable for small-scale or fast-prototype applications.

FPGA

A special case of the gate-array layout is the *Field-Programmable Gate-Array*, or FPGA, topology. In this topology, the chip is bought pre-fabricated and packaged. The feature that makes FPGAs stand out among gate-array topologies is that, instead of effecting a design with a photo-mask, all wires and interconnections are manufactured on the chip, and programmable *fuses* are fabricated into the interconnections. The designer can effect a design by programming the interconnections between the gates and the wires, altering the connectivity between the pre-fabricated logic blocks in the array. This programming requires minimal hardware, and no fabrication, so the designer can physically implement a design in-house in a short period of time, hence the term "field-programmable".

Individual FPGA chips can be quite expensive. Large state-of-the-art chips can cost thousands of dollars each, making them impractical for anything but the smallest production runs. FPGAs are also not very space-efficient, since wires and interconnects are purposely generic to allow a variety of uses. Also, because these circuit use fuse-technology, which add a significant delay to interconnections, they are not suitable for very speed-demanding or low-power applications.

However, because the chip topology is highly regular, effective automated design tools exist that can allow a fast design turnaround with a minimum of human interaction, minimizing design-time costs. Today's FPGAs can contain millions of transistors and can implement an entire hardware "system" in a single chip (called Systems On Chip, or SOC). And because all but the most inexpensive FPGAs are re-programmable, they are good for testing incremental changes to a design *in hardware*. Therefore, the FPGA is very well-suited for fast-prototyping a design in hardware.

Standard-Cell

In between full-custom and FPGA circuits are several intermediate topologies that provide a compromise between good design time and production size. These topologies can be loosely termed the Application-Specific Integrated Circuit, or ASIC, topologies. FPGAs are sometimes included in this category, but are usually considered to be used for too small a production to be a

true ASIC. FPGAs are often used to prototype other ASIC designs, however. The exact definition of an ASIC is highly debatable, but is usually considered to be any partially pre-fabricated chip that requires further fabrication to complete a design, targeted for a small- to medium-production run, and designed for a specific task or set of tasks. Examples of ASICs include a video controller chip, or an encoder chip in a television remote-control. A micro-controller or an FPGA is usually not considered an ASIC because of its flexibility of design. Both are fabricated in order to fill a number of roles, and can be programmed for specific tasks, but are not designed for any one task in particular. Of course this is a pretty fine line, and one or both can be argued to belong to the family of ASICs, but for this discussion they will not be included.

The *Standard-Cell* topology is one in which the logical circuit design is first mapped to a library of standard-cells. The designer can increase productivity by utilizing a pre-designed standard-cell library. The library typically includes common logic gates and small functional blocks such as multiplexers and decoders, and sometimes even small storage elements such as latches or flip-flops. All standard cells in a library are restricted to having the same height, but their width can be chosen by the standard-cell library designer to accommodate the area of the functional block design. Once a circuit is mapped, the circuit of standard-cells is laid out in rows within the chip boundaries. Spaces between the rows, called channels, are used to implement the interconnections between standard-cells. I/O connections are placed at the edge of the chip. The standard-cell topology is illustrated in Figure 2.1(B).

The particular standard-cells used are chosen by a designer to implement a design. Therefore the topology is flexible enough to efficiently implement many designs. However, because the topology has a great deal of structure, it is also well-suited for automated design. The design tool's job is to lay out the cells in rows such that the interconnection length is minimized, and the chip area is minimized. However, the variable-width aspect causes complications in automation, and the final result must be fully fabricated, so it is best suited for medium-sized production where area or performance is a major consideration.

General-Cell

The final design style that will be mentioned is the general-cell, or macro-cell, topology. This topology is made up of a (usually) small number of irregularly-shaped blocks, with interconnections being laid down in the spaces between the blocks, as in Figure 2.1(C). The irregularly-shaped blocks introduce several design challenges, but the small problem size makes it suitable for computationally demanding solution methods. These methods can arrive at high quality solutions in very short amounts of time.

2.3 Standard-Cell Placement

The work in this thesis presents two new methods for handling the highly complex task of designing ASIC chips. The usefulness of these methods is demonstrated by applying them to the design task of cell placement for standard-cell topologies. These methods are general enough that they would also be suitable for other layout styles such as FPGAs and gate-arrays, with suitable modification.

The standard cell placement problem is the problem of arranging a circuit of interconnected equal-height, variable-width, "standard cells" into parallel rows (see Figure 2.1) such that the total interconnection length, placement area, or some other performance metric is minimized. The standard-cell placement problem has been shown to belong to the NP-complete class of problems, so heuristic solution methods must be used [GarJoh79].

2.3.1 Problem Overview

The placement problem, as it pertains to this report, can be described as follows: given a set of modules and nets, assign the modules to legal positions within a placement area such that the interconnection cost between the modules is minimized.

The set of modules in the network is denoted by M and the set of nets by N. The set of modules can be divided into cells, which implement logic and are placed in the chip core region, and pads, which provide external connection and are placed around the periphery of the chip. The modules connected by a net $n \in N$ is denoted by M_n and the set of nets connected to a

module $m \in M$ is denoted by N_m.

In the standard-cell layout, cell modules are placed within R parallel rows in the chip core area such that no cells in a row are overlapping, and a maximum row length is not exceeded. Pad modules are placed at the placement area's periphery. In the placement stage of design, the actual physical placement of interconnections is not known, but is approximated with measures such as HPWL (described in the next section). In the global and detailed routing stages, interconnections are placed in the inter-row gaps between rows and between rows and pads.

2.3.2 Interconnection Cost

It is too computationally expensive to determine routes for wires to take to interconnect cells and pads during the placement design stage. Because of this, the routing cost is approximated during placement, and a separate routing design step is performed after placement. The most commonly used estimate of interconnection cost is the half-perimeter of the smallest bounding box containing all pins of a net. This is called the half-perimeter wirelength (HPWL), and is illustrated in Figure 2.2.

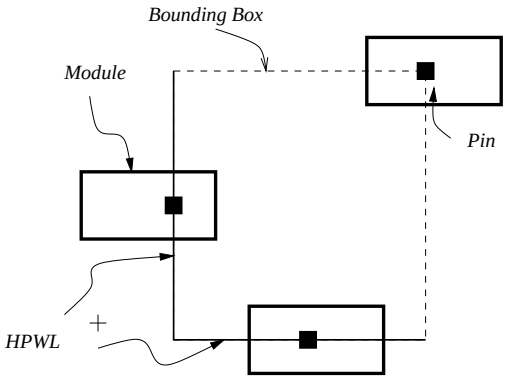

Figure 2.2: Half-perimeter Wirelength is an approximation of the routing cost of a net

The HPWL of a net approximates the length of a minimal Steiner tree, which is a lower bound on the final routing cost of a net. By increasing the rout-ability of the circuit, we can reduce the overall area of the final layout.

Given that a cell m has center coordinates (x_m, y_m), and a cell's pin connecting to net n has coordinates relative to the cell's center (ξ_{mn}, η_{mn}), the pin's absolute coordinates are $x_{mn} = x_m + \xi_{mn}$ and $y_{mn} = y_m + \eta_{mn}$. The half perimeter of net n is calculated with the following equation.

$$HPWL_n = \max_{m \in M_n} \{x_{mn}\} - \min_{m \in M_n} \{x_{mn}\} + \max_{m \in M_n} \{y_{mn}\} - \min_{m \in M_n} \{y_{mn}\} \tag{2.1}$$

2.3.3 Placement Quality Measure

The most common objective for the placement problem is to minimize the routing cost. Routing cost is used because reducing it actually minimizes a number complimentary design goals. By reducing the routing length, the area needed by the interconnections (and therefore, the chip) is reduced, and more chips can be manufactured on the same chip wafer, increasing chip yield and lowering cost [SaiYou95]. Shorter interconnections also result in an increase of chip speed due to the reduction in interconnection capacitance [BelElm95]. Power reduction is also reduced due to the same reduction in capacitance [WesEsh93].

Therefore, the basic placement objective function is to minimize the total half-perimeter wire-length for all nets in a placement, since the HPWL of a circuit is an estimate of the final routing cost (see the previous section):

$$\text{Minimize} \sum_{n \in N} HPWL_n \tag{2.2}$$

There are numerous other terms that can be added to the objective function to directly optimize the various design goals of section 1.3, such as minimizing the length of a *critical path* to meet timing constraints, or to minimize the final area of the chip by minimizing the longest standard-cell row length.

Minimizing several terms in an objective function is possible, but it can be very difficult to strike a balance between competing goals, and can be computationally demanding. More commonly, *constraints*, or acceptable limits, are placed on key parameters, and any solution that meets those constraints is permissible.

In the standard-cell problem, there are constraints on the allowable values of x and y co-ordinates that are imposed by the standard cell layout style for semi-custom chips. The x and y co-ordinates for a pad module must represent a position on the edge of the standard cell layout area. The value of x and y for cell modules must be such that the entire cell is within a row. As well, no cell or pad may overlap with another.

Other objectives can be met by either objective function terms, or by fixed constraints. For example, in standard-cell layout, the row-lengths should be approximately equal to minimize the chip width. This can be met by either inserting a large-row penalty term directly into the objective function, or by placing constraints on maximum row lengths.

One important objective that should be mentioned is that the placement algorithm should lead to a routable chip, i.e., a chip that can be connected subject to the constraints of the implementation technology. This objective is not normally reflected in the objective function of the placement phase, due to the high complexity of the routing task [Are95]. Instead, the chip is placed first, and then an attempt is made to route it. If it is not routable, the placement phase is repeated until the routing is successful.

2.3.4 Placement Problem and the Traditional Quadratic Measure

As an example of a method of approximating a solution to equation (2.2), a mathematical formulation of an *approximation to* the standard-cell placement problem is now presented. A circuit is represented by a hypergraph $G(V, H)$, where the vertex set $V = \{v_1, v_2, \cdots, v_n\}$ represent the nodes of the hypergraph (set of cells to be placed), and $H = \{e_1, e_2, \cdots, e_m\}$ represents the set of hyperedges of the hypergraph (set of nets connecting the cells). The two dimensional placement region is represented as an array of legal placement locations. Ultimately, the placement task seeks to assign all cells of the circuit to legal locations such that cells do not overlap. The hypergraph is transformed into a graph (a hypergraph with all hyperedge sizes equal to 2) via clique model for each net. Each edge e_j is an unordered pair of vertices with a nonnegative weight w_j assigned to it. Each cell i is assigned a location (x_i, y_i) on the XY-plane. The cost of an edge connecting two cells i and j with locations (x_i, y_i) and (x_j, y_j) is computed as the product of the squared l_2

norm of the difference vector $(x_i - x_j, y_i - y_j)$ and the weight of the connecting edge w_{ij}. The total cost, denoted $\phi(x, y)$, can then be given as the sum of the cost over all edges; i.e:

$$\phi(x, y) = \sum_{1 \leq i < j \leq N} w_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2] \tag{2.3}$$

This formulation does not actually restrict cells to occupy legal positions, and so minimizing (2.3) produces a placement with a great amount of overlap among the cells because cells sharing common nets are attracted together. However, this formulation can be solved in polynomial time (i.e., it is not an NP-complete problem) [Eta99]. We refer to formulation (2.3) as the QP (Quadratic Programming) formulation.

2.4 Approaches for Solving the Standard-Cell Problem

Standard-cell placement is a NP-complete combinatorial optimization problem. Many techniques have been developed to attempt to find the optimal solution in a reasonable amount of time. Figure 2.3 shows a taxonomy of the most widely-used approaches to solving a physical design layout problem.

The most basic division among approaches is between exact solution methods and approximation methods, also called *heuristics*. Exact solution methods are precisely that - they solve the problem formulation of section 2.3.3 exactly, giving the optimal solution. In reality, exactly solving an NP-complete problem, like standard-cell placement, takes an extraordinary amount of time, and so these methods are of little practical use. Approximation methods, on the other hand, give sub-optimal solutions, but can be implemented to operate in reasonable amounts of time. Every method used industrially is of this type, since design time is of great importance in industry‘. The measure of the quality of an approximation method is the distance of the solution from the global optimal solution.

Approximation methods can be broadly divided into three classes of approaches: global methods, constructive methods, and iterative improvement (search) methods. In global methods, cells are placed in globally-good positions by relaxing the constraints of the problem. Constructive

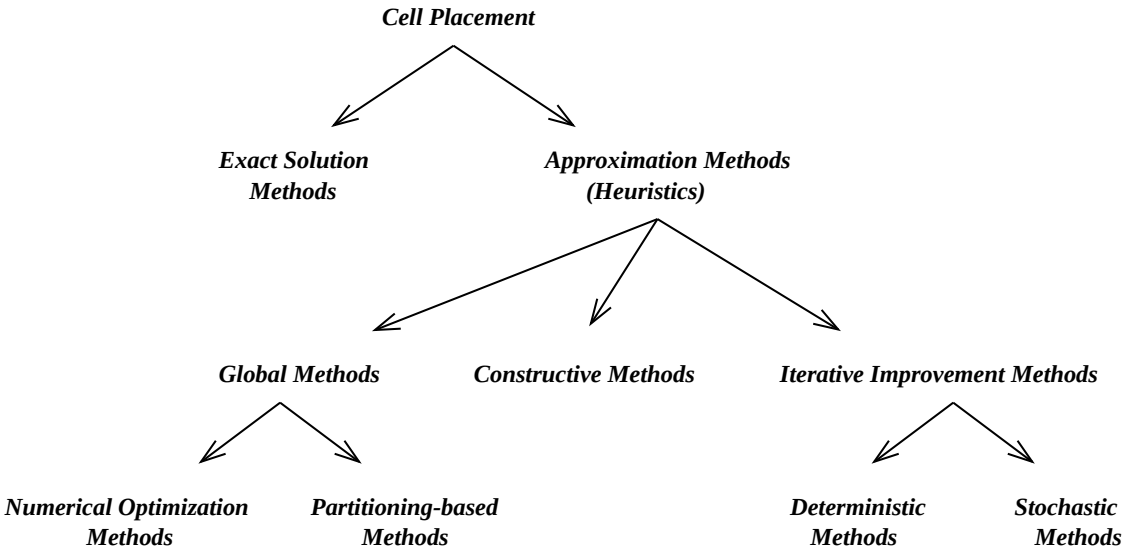

Figure 2.3: Taxonomy of Solution Approaches to Layout Problems

placement methods start with a small set of cells and "build" a placement by iteratively adding unplaced cells adjacent to already-placed cells. Iterative improvement methods start with a legal placement and improve it by searching for small perturbations to the placement that result in better solutions. The Approximation methods are discussed in more detail in the following subsections.

2.4.1 Numerical Optimization

One global placement approach is numerical optimization. In these methods, the standard cell problem is formulated in terms of an objective function to be minimized (or maximized) subject to a set of constraints. The formulation is then solved exactly using mathematical programming techniques such as linear, non-linear, integer, and dynamic programming techniques [Eta99, Beh99].

This appears to be the same approach as an exact solution method, which was already shown to be unsolvable in reasonable time. However, in numerical optimization methods, the original problem is approximated by a similar problem that **can** be solved in polynomial time. The global optimum to this new problem is then found using modern mathematical programming techniques,

and is used as an approximate solution to the original standard-cell problem. If the approximation model is good, solution quality obtained in this way can be very good. An example of a numerical optimization formulation is the QP formulation presented earlier in section 2.3.4, in which the rectilinear wirelength measure is approximated with a quadratic "straight-line" measure, which is much more easily solved.

The solutions arrived at in this manner are good from a "global" perspective, since they are optimal according to a mathematical model of the system, but are sub-optimal according the actual chip layout. This is because the cell positions attained are not actually legal solutions (remember, in standard-cell layout cell positions are constrained to non-overlapping positions in a row). This is often referred to as a "top-down" method, since the global view is considered before the detailed view is considered. To get a legal solution, a legalization heuristic must be used to find a "good" legal position for each cell. A good legalizer minimizes the difference in wirelength (or other objectives) between the global solution and the legalized solution.

The quality of the solutions is partially a factor of the mathematical model used to represent the placement problem, and partially a factor of the legalization method used. Many methods have been used to approximate the exact standard-cell problem, the most popular being linear models [KleSig91] and quadratic programming models [CheKuh84, Beh99, EtaAre99].

Global placers are never used alone. Because of the error introduced by the legalization process, solutions are sub-optimal compared to search techniques. However, search techniques can be used *after* global placement to further improve the quality.

2.4.2 Min-Cut Placement

Another popular global placement approach is min-cut placement (e.g., [HuaKah97]). In these methods, a circuit is recursively bisected (partitioned into two parts), minimizing the number of nets that connect components between partitions (i.e. minimizing the "cuts"). High-performance partitioning itself is usually performed using search heuristics (discussed in the next sub-section).

Min-cut placement achieves high-quality placements for two reasons. First, the heuristic behaves as a top-down method in early iterations, placing cells in the general regions that they

should belong to. As the partition size decreases, it behaves more as a bottom-up method, placing cells in smaller and smaller regions, eventually reaching a partition size of a single cell. Second, it implicitly reduces net congestion as a function of min-cut partitioning. This not only increases the chances of achieving a routable circuit, but tends to reduced the initial wirelength.

Like numerical optimization methods, partitioning-based methods do not directly attempt to minimize wirelength (or other common objectives), and so the solution obtained is sub-optimal in terms of wirelength. Search heuristics are used to further improve the solution after min-cut is used.

2.4.3 Constructive Placement

In this method, a placement is constructed by choosing a set of *seed* cells which are initially placed in the placement area. Then, recursively, a set of cells are placed adjacent to the seeds, until the placement has "grown" from the original seed cells.

Constructive placement is a "bottom-up" method, since it considers only the local environment of each individual cell, placing each cell at the best location available at the moment. In this method, cells tend to be placed well locally, but have poor overall quality.

Because of their lack of global perspective, constructive methods do not provide high quality solutions. However, they are usually very fast, and can therefore be used as a starting point for search-based heuristics, instead of a purely random initial solution.

2.4.4 Iterative Improvement (Search Heuristics)

A search heuristic begins with an initial (legal) solution, and attempts to improve it by incrementally searching the solution space. Better solutions are obtained by perturbing the solution in some way such that the new solution has an "acceptable" cost. We do not say a "lower" cost, since stochastic search heuristics find good solutions by occasionally accepting higher-cost solutions to escape local minima in the search space.

The goal of any search heuristic is to find the *global optimum*, which is the best possible solution to the problem. For an NP problem, however, it is not possible to exhaustively search

all possible solutions and take the best. Instead, search heuristics attempt to find a good *local optimum*. Eventually during a search, given enough time, a solution is found where no single perturbation improves the placement quality. This situation is called a local optimum. The best possible local optimum is also the global optimum.

A *deterministic*, or "greedy", heuristic only accepts the best possible perturbation at any time. Deterministic heuristics are fast, but they also get trapped in local minima quickly. When applied to NP problems, this usually means that greedy heuristics find very sub-optimal solutions.

Stochastic heuristics use some "randomness" to accept some poor solutions, which allows the search to escape some local minima when encountered. For a stochastic search, it may be possible to accept several poor solutions in order to get to a point where moves again improve the solution quality, hopefully to a new local minimum that is better than the previous local optimum. Heuristics that are able to escape local optima by accepting some poor solutions are said to exhibit "hill-climbing" ability.

In the following subsections, the most common approaches to iterative improvement are presented as examples.

Breadth-First Search

An example of a simple search method is breadth-first search. Each cell in a placement is moved in turn to a new location, and the solution is accepted if the cost is reduced. It is "breadth-first" in that each cell is attempted exactly once before a second attempt is made. Breadth-first search is also a *greedy* method, since it only accepts moves that reduce the total cost.

Simulated Annealing

An example of an advanced search heuristic is simulated annealing [KirGel83]. Simulated annealing is a mathematical analogy to the annealing process in chemical engineering, where crystals are grown in a hot metal by carefully controlling the rate of cooling of the metal. In the placement problem, simulated annealing is used to "crystallize" a good placement by controlling the rate of improving and non-improving cost changes as a placement is perturbed. Simulated annealing is a

stochastic method with hill-climbing ability, since it can accept a series of poor moves to escape a locally-optimal solution. Many implementations of simulated annealing have been applied to the standard-cell problem (e.g., [Gro86, SecLee87, MalGro89, SunSec95]).

In simulated annealing, a cell is chosen at random, and either moved to a new location, or swapped with a neighboring cell. If the cost of the solution is reduced, the move is accepted. However, if the cost is increased, the move may still be accepted with a probability of $e^{-\Delta C/T}$, where ΔC is the change in cost, and T is the *annealing temperature*. Initially, T is set to a very high value such that most moves are accepted. As moves are accepted, T is gradually decreased, lowering the number of poor solutions that are accepted. After many moves have been accepted, T is very low, and only improving solutions are accepted.

The rate of temperature change is called the *annealing schedule*, and determines the final solution quality obtained. Simulated annealing has been mathematically proven to converge to the global optimal solution when used for a long-enough time with a good annealing schedule [MitRom86], but this time is too long for use for practical use, and some good local optimum is usually accepted as a final solution.

Force-directed Improvement

One method that was developed for placement problems is force-directed improvement [Ger99]. In this search method, the nets are considered to exert a pulling force on all connected cells. When a cell is considered for a move, all these forces are considered, and the cell is moved to the location nearest it's center-of-potential. This method can be implemented to run quickly, and has shown to perform well. However, determining the weight function for each net is difficult, and varies from circuit to circuit.

2.5 Test Circuits

Tables 2.1 and 2.2 show the benchmark circuits used to measure the performance of the heuristics in this thesis. The circuits used were the MCNC '91 benchmarks [Koz91]. This test set consisted

Circuit	Cells	Pads	Nets	Pins	Rows	Pad Distribution			
						Top	Bottom	Left	Right
Fract	125	24	147	462	6	22	2	0	0
Primary1	752	81	904	5526	16	21	20	20	20
Struct	1888	64	1920	5471	21	64	0	0	0
Industry1	2271	814	2478	8513	15	254	258	302	0
Primary2	2907	107	3029	18407	28	30	16	30	31
Biomed	6471	97	5742	26947	46	8	72	9	8
Industry2	12142	495	13419	125555	72	107	126	123	139
Industry3	15059	374	21940	176584	54	113	124	63	74
Avq.small	21854	64	22124	82601	80	30	34	0	0
Avq.large	25114	64	25384	82751	86	30	34	0	0

Table 2.1: Benchmarks used for testing

Circuit	Cell Width				Module Degree			Net Size		
	Min	Max	$\bar{x}$	σ	Max	$\bar{x}$	σ	Max	$\bar{x}$	σ
Fract	16	64	31.1	16.8	7	3.5	1.49	17	3.14	2.25
Primary1	30	200	106.6	70.1	9	3.7	1.15	18	3.26	2.60
Struct	16	40	26.0	7.8	4	2.9	0.6	17	2.84	1.90
Industry1	12	84	31.6	11.8	9	3.5	1.0	320	3.47	8.55
Primary2	30	300	86.3	57.7	9	3.7	1.5	37	3.64	3.76
Biomed	16	72	35.1	18.6	6	3.2	1.0	43	3.05	3.40
Industry2	32	280	82.3	41.6	12	3.9	1.7	427	3.46	7.47
Industry3	56	232	93.6	32.9	12	4.4	1.5	325	3.00	3.23
Avq.small	16	72	33.1	19.6	7	3.5	1.4	451	2.64	4.47
Avq.large	16	72	31.9	18.5	7	3.3	1.3	451	2.55	4.18

Table 2.2: Connectivity and Cell Sizes of Benchmarks

of ten circuits ranging in size from 125 cells to over 25000 cells.

Table 2.1 shows the net list statistics for each benchmark used. The pad distributions indicated the number and location of pads, and the rows indicates the number of standard-cell rows in which to place the circuit. Pad distribution and number of rows are parameters dictated by the benchmark circuit files.

Table 2.2 shows the distribution of cell sizes, cell connectivity, and net connectivity. The minimum module and net degrees are not given since they are always equal to '2'.

The circuits have been grouped according to size: small, medium, and large, indicated by horizontal lines in Table 2.1, where the "small" circuits are Fract, Primary1, and Struct, the

"medium" circuits are Industry1, Primary2, and Biomed, and the "large" circuits are Industry2, Industry3, Avq.small, and Avq.large. These groupings by size will be indicated in all further tables of results, and should be considered when observations or conclusions are made for "small", "medium", and "large" problems are made. The primary basis of this division is the number of pins in the circuit, since the number of pins determines in a large part the time needed for improvement, although it also corresponds to the number of modules or nets in the circuit as well. The last four circuits are generally recognized as "large" problems. These circuits are the primary focus of this thesis. The other benchmarks are presented and used for completeness, but are not as valuable for drawing conclusions since the problem size is such that computationally complex placement techniques can be used on them in reasonable amounts of time. Since the focus of this thesis is to present fast heuristics for use on large problems, it is reasonable to expect that slower, more exhaustive techniques should perform better on smaller circuits.

2.6 Summary

The VLSI design process is broken down into simpler, more computationally tractable tasks, to manage the high complexity of the task. One of these tasks is physical design, which, while a much simpler task when handled independently, is still incredibly complex. Not surprisingly, this complexity is handled by dividing the physical design task into more tractable sub-tasks.

Many different circuit topologies exist to implement a circuit, such as full-custom, gate-array, and standard-cell layouts. Each of these topologies imposes design constraints and optimization concerns, which affect the design of a circuit.

The layout topology used to demonstrate the search and complexity-management techniques in this thesis is the standard-cell layout style. This style has a number of unique characteristics that make it particularly difficult to automate, and so is an ideal candidate for demonstrating these new approaches.

Many approaches have been taken to solve the standard-cell problem, and ASIC design problems in general, including global, constructive, and iterative improvement methods. Finding the

absolute best solution to a large problem is virtually impossible, so all practical solution approaches are categorized as *heuristics*, and therefore find sub-optimal solutions.

It is difficult to measure the quality of a solution approach, since ASIC design is computationally very complex and usually an optimal answer is not known by which to compare results. To overcome this problem, a standard set of benchmark circuits are used to compare behavior and solution quality obtained with other solution technique. Any new technique should compare results on this basis.

In the following two chapters, two new approaches to dealing with the high complexity of ASIC design are presented. In the next chapter, a deterministic search heuristic is presented and applied to iterative improvement for the standard-cell placement problem. In chapter 4, a clustering heuristic is used to take a different approach to ASIC design, by reducing the complexity of the problem.

Chapter 3

Utility Function-Based Iterative Search

3.1 Introduction

Iterative improvement methods seek to improve an initial problem solution by making small changes, such as moving or swapping individual cells in the placement problem. Iterative improvement search techniques are applicable to many different fields of study. In the remainder of this chapter, we will focus on the application of iterative improvement to the standard-cell placement problem.

In standard-cell placement, new states are created by moving a cell or set of cells to new locations. Possible movements to examine are usually selected randomly. This randomness of selection can be advantageous to a search heuristic, since it can allow the search to escape local minima (see section 2.4.4). This is desirable early-on in the improvement process, when the solution is far from a local minima, and a good search heuristic is attempting to "guide" the search to a good minimum. In later design stages, such as after a good global placer has been used and the solution is already close to a good local minima, all that remains is to seek out the local minimum as fast as possible. However, the search space in a typical placement problem is

enormous. Late in a search, very few possible moves actually improve the quality of the solution. A purely random move selection can cause the search to spend a disproportional amount of time examining poor moves, and the move *acceptance rate* will be low. What is desired is greedy search heuristic that reduce the number of possible moves to examine, and thus speed up the search process by increasing the acceptance rate.

In this chapter we examine a novel new approach to overcome this problem, by utilizing a measure of a cell's placement quality to choose moves that are statistically more likely to improve. *Utility*, in the scope of the placement problem, is a measure of the quality of a current placement relative to its optimal placement. A simple iterative interchange heuristic can be implemented, selecting cells for possible moves based on the utility value of those cells. By concentrating computational effort on cells with worse-than-average placements, greater gain can be achieved with less effort than that which a randomized method could achieve in the same time. In this way, it is possible to reduce the computational impact of examining moves that will be rejected, while increasing the acceptance rate versus a comparable method choosing moves randomly.

In section 3.2 previous attempts to overcome this problem are presented. Then, in section 3.3 a new method based on utility theory is presented. Section 3.4 shows the results of using the utility-based improver as part of a state-of-the-art standard-cell package. Finally, in section 3.5 we discuss the effectiveness and usage of the new improvement scheme, and propose further directions for work.

3.2 Previous Work

It was recognized long ago that a deterministic move selection mechanism (i.e. not purely random) could significantly speedup a search. Simulated Evolution (SE) is one method proposed to overcome this problem [KinBan89]. In SE, the cells to be moved are chosen deterministically, using a selection criterion the authors called the "goodness" of a cell. In a pass of the search heuristic, a threshold goodness value is randomly selected. Cells with a goodness the threshold value are marked for relocation. Then, these cells are simultaneously removed from the layout area

and then placed by solving a matching problem between marked cells and free placement spaces. Results using SE proved that deterministic selection of possible moves does achieve a fast gain, and the selection can be done with little overhead compared to a random selection mechanism. However, the actual mechanism for the calculation of goodness in SE is very unsophisticated, and the matching method of placement in the SE heuristic is too slow for practical implementation.

Another method for reducing the number of rejected states is to select candidates for a move based on the size of cells. If a candidate cell is moved to a new row, and the length of the row would exceed row length limits, the candidate cell is discounted. This is now a commonplace heuristic feature (e.g., [SunSec95]), and can be incorporated into practically any selection mechanism. The heuristic presented in this chapter restricts move selections based on cell sizes.

A method to increase the acceptance rate of moves is to use a *local search window*. A local search window is a region surrounding a candidate cell for movement. Allowable positions to move the cell, or to select a partner for a location swap, are limited to the window, thus keeping searches localized to the candidate cell. The motivation for this is the observation that in a global placer, or some other search method that places a circuit from a top-down perspective (see section 2.4), modules are already located in a globally good, but locally sub-optimal, placement. By limiting the range of possible moves, the likelihood that these final locations will be found, and speed of convergence to the corresponding local minima, are increased. Using a search window has been shown to greatly increase the convergence rate of a search, and finds good local optima when starting from a good *global* initial placement [Ken97, SunSec95]. When used on a poor initial placement, the local search window imposes too great a restriction on allowable moves, and the final solution quality is poor. The heuristic we develop in this chapter uses the concept of a local search window to increase the acceptance rate of moves.

A different method for the selection of moves proposed in this chapter is based on utility theory [EtaVan98]. Utility theory has been used in physical design automation in the past for the channel routing problem [EtaVan98]. The concept of utility is very similar to the concept of goodness in simulated evolution, but where the goodness of a cell is a measure of "survivability" from one pass to the next, utility is a measure of "room to improve", or ideally, "distance from

optimality". This allows utility to be more flexible, since it is not a simple binary survive/not survive decision. There are other implementation differences as well, making SE and our heuristic distinct approaches to the improvement problem.

3.3 Solution Methodology

3.3.1 Utility of Net Placement

The utility of a net is a measure of how close to optimality the routing cost of a net is, where the routing cost is approximated by the net's HPWL. We calculate the net utility as the ratio of a net's current wirelength to its ideal wirelength. The current wirelength of a net can be estimated with its HPWL. However, knowing the ideal wirelength of a net is equivalent to knowing the ideal placement of its connected cells! Since this is just a restatement of the placement problem itself, clearly we can't calculate the ideal wirelength of a net. So what can we do?

We don't know the ideal position of all cells in a placement (since this is precisely the problem we are approximating), and therefore, we can't know the ideal wirelength of any net in such a placement. However, if we focus our attention on a single net and disregard all other nets in the circuit, then it obvious that the best placement for those cells connected to the net is if all the cells are adjacent to each other, since this will result in the smallest wirelength for the net.

It turns out, however, that even determining the optimal placement for cells connected to a single net is not trivial for the standard-cell layout style. In a more regularly-structured design layout style, such as FPGAs, a ideal relative placement can be found in constant-time. For standard-cells, it is a problem of $O(n!)$ complexity for standard-cells, since the width of a cell and the pin placement on the cells is variable - a big difference! To make this heuristic more practical, an approximation on this bound must be made. The approximation used described in detail in later in section 3.3.2.

Assuming we have calculated the current wirelength and the wirelength lower-bound, the

utility U_{net} of net i is calculated as

$$U_{net,i} = \frac{BOUND_i}{HPWL_i} \tag{3.1}$$

Where $BOUND_i$ is a close approximation of a lower bound of the HPWL of net i, calculated by the method in section 3.3.2, and $HPWL_i$ is the half-perimeter wirelength of net i, described in section 2.3.2.

3.3.2 Wirelength Lower Bound

It was shown in the previous section that calculating a net's wire length lower bound is an NP problem. However, we can readily approximate the cells' *relative* placement to each other. The algorithm used to approximate a lower bound on a net is similar to that used in [KinBan89], and is illustrated in Figure 3.1.

As a first approximation of the optimal placement, all cells of the current net are assumed to be placed next to each other without space between them, as though they were all positioned in a single row (Figure 3.1(b)). Next, all cell areas are collapsed to an approximately square region (since a square has the minimal perimeter among all rectangles of the same area). Row spacing is considered and included in the computation. Figures 3.1(c) and (d) show the result after a net has been collapsed over two and three rows, respectively. Note that in Figure 3.1(c), the region is closer to a square shape (as measured by its aspect ratio) than that in Figure 3.1(d), so this configuration would be accepted.

This leads to a minimum perimeter considering only the cell dimensions. However, the minimum bounding rectangle (from which the HPWL is derived) depends on the relative position of the pins which connect the cells to the current net. Recall that pin positions are actually approximated as a position somewhere *inside* of a cell (e.g. as in Figure 3.1(a)).

To estimate the effect on the actual net length, the lower bound is *approximated* by subtracting half of the row height from the top and bottom sides, and 3/4 of the width of an average cell of the net on the left and right sides of the region. The half-perimeter of the final (dashed) region

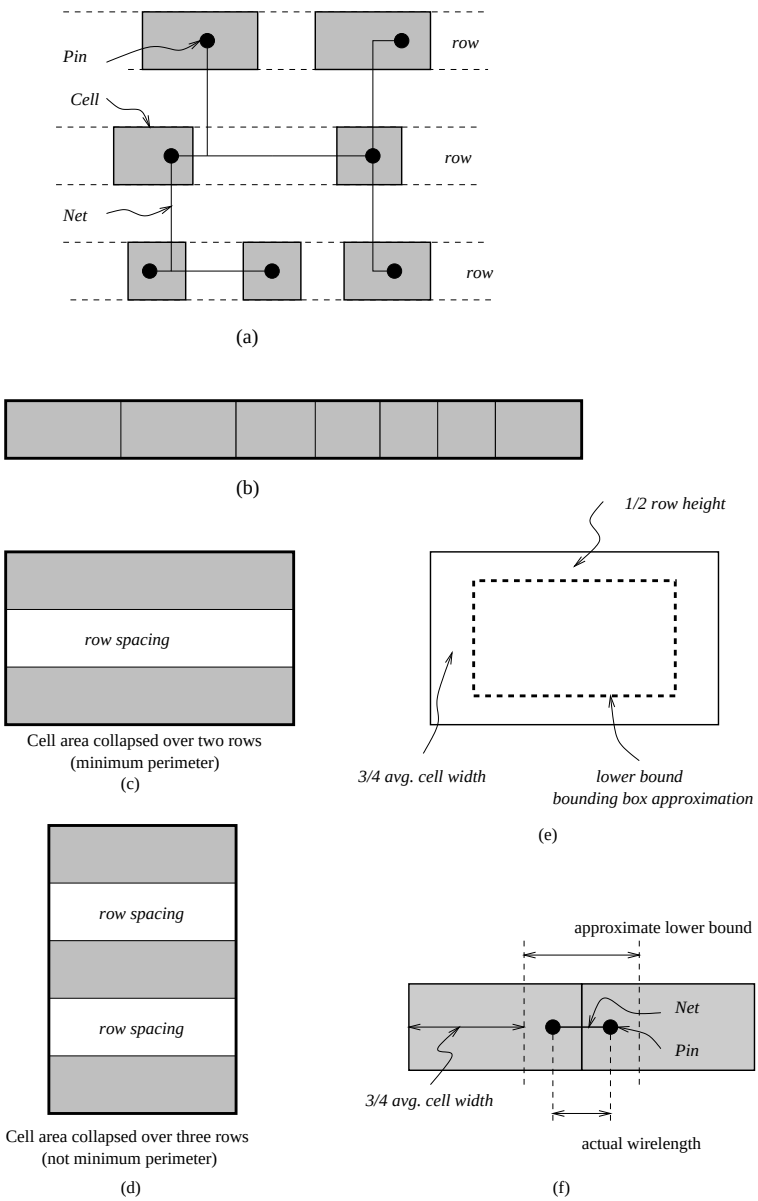

Figure 3.1: wirelength Bound Computation

of Figure 3.1(e) is then taken as the lower bound.

This bound is not a true lower bound, since it is possible for a placement of cells to have a wirelength lower than this value. This situation occurs for small nets that have all pins situated farther than 3/4 of an average cell width from the outside of the boundary region, as in Figure 3.1(f). However, this situation only arises when a net's placement is locally optimal or very nearly optimal, so we reflect this information by changing the utility of these nets to a utility of '1.0' - the true optimal utility. Larger nets are rarely super-optimal, since the probability of a net being optimally placed decreases rapidly with net size.

Calculating the lower bound in computationally expensive, since we must iterate over several rows to find the optimal collapsed area region for a net to occupy. However, calculation of the lower bound is independent of an actual circuit placement, and so the lower bound can be calculated for all nets as a pre-processing stage before a search is begun. During a search, a net's utility can be quickly calculated at any time by dividing the current HPWL of the net by its lower bound.

3.3.3 Cell Utility

Cell utility is a measure of a cell's placement quality based on the placement of the nets it is connected to. The utility of cells in a placement is used to select cells for movement, so the goal of calculating cell utility is to reflect the degree to which a cell's placement can improve.

It is very difficult to significantly improve large nets. Notice that on a two cell net (Figure 3.2(a)), the perimeter of the net (and therefore the HPWL measure of placement cost) is always due to the placement of the two cells on the net, i.e., HPWL can be improved by moving either cell closer to the other. In a larger net, such as the six cell example in Figure 3.2(b), the placement can be *changed* by moving any cell, but it can only be *improved* by moving one of the cells on the net's perimeter. Obviously, as the net size grows (and in large benchmarks, the largest nets connect hundreds or thousands of cells), the chances of selecting one of the perimeter cells *and* moving it in the correct direction diminish rapidly with increasing net size.

While only a few cells in a typical net can reduce the perimeter, moving *any* cell in a net can increase the net perimeter. It is obvious that the size of a net will usually be directly related to

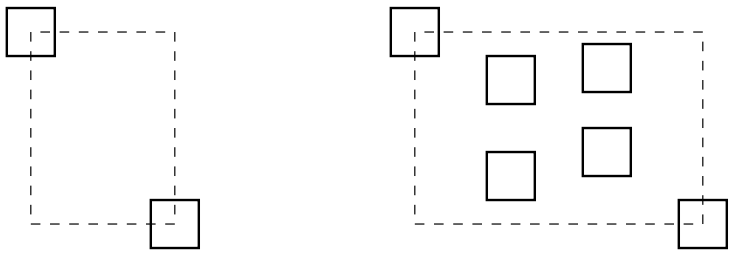

Figure 3.2: Large nets are difficult to Improve

the size of its perimeter, but is inversely related to it's ability to improve. In other words, the larger the net, the harder it is to make its placement better. If this effect is not accounted for appropriately in a measure of cell utility, it could give a misleading indication of a cell's ability to improve, particularly if it is connected to several very large nets. A good measure of cell utility should therefore consider the size of a net.

There are several ways to reflect the size of a net when calculating the cell utility. One method would be to consider the size of nets when calculating *net* utility. However, this would require extra computation time to calculate utility for those large nets that, as just described, are very unlikely to be improved. Another method would be to weight all net utilities in the cell utility calculation, but again, this would still entail calculating all net utilities for nets attached to the cell. *Because large nets are difficult to improve, large nets should not be used in cell utility calculations!* The actual cutoff value is discussed later.

The method used in the utility-based heuristic for calculating cell utility is therefore to average all connected nets' utilities, *disregarding large nets.* That is

$$U_{cell} = \frac{\sum_{i \in N_m} U_{net,i}}{|N_m|} \tag{3.2}$$

where the size of net i is smaller than some user-specified threshold size, and U_n is the net utility, discussed in section 3.3.1. If large nets are disregarded, the utilities for these nets does not need to be calculated and maintained.

Selecting the actual net size cutoff value is not trivial, because of the variations in benchmark circuit statistics and the interactions between cell and net utility and gain estimation. As Figure 3.3 shows, the vast majority of nets connect between 2 and 5 cells, so a cutoff should definitely be higher than 5 cells. Move gain estimation (discussed later in section 3.3.7 considers only nets below 30 cells in size. The consequence of this for cell utility is that, if the utility of a particular cell is dominated by larger nets, and the move actually improves the placement, gain estimation will not reflect this fact so the move could be rejected.

A threshold value of 30 cells was used, since over 99% of the nets in the largest benchmarks are smaller than 30 cells, and gain estimation uses a 30-cell threshold. Furthermore, disregarding large nets has been shown to greatly reduce run-times during other aspects of physical design in large circuits, such as cost estimation, without adversely affecting solution quality [SecLee87, Are95]. These assumptions are investigated in section 3.4.4.

In the utility-based heuristic, cell utilities are not maintained *within* a pass due to the high computational cost of maintaining a sorted utility list, and the inaccuracy introduced by using estimated net wirelengths (see section 3.3.7 within a pass. At the beginning of a pass, the cell utilities are calculated for all cells and sorted. This sorted cell utility list is then used to "suggest" cells that have a high potential for wirelength improvement. Note that a larger pass size will lead to decreasing utility accuracy over time within a pass.

3.3.4 The UTILITY Heuristic

Iterative improvement approaches typically choose cells randomly. What makes the utility-based approach novel is the mechanism it employs to select possible cell movements. The utility-based iterative improvement heuristic (UTILITY) is shown in Figure 3.4.

The calculation of net and cell utilities in line 1 of Figure 3.4 is performed as in the previous sections. In line 4, a percentage of the sorted utility list, called the *search depth*, is examined. Starting at the lowest utility value (corresponding to the worst-placed cell), cells are examined, one per pass, and removed from the utility list until the search depth has been reached. The local search window and target selection of lines 6-13 are discussed in sections 3.3.5 and 3.3.6, below.

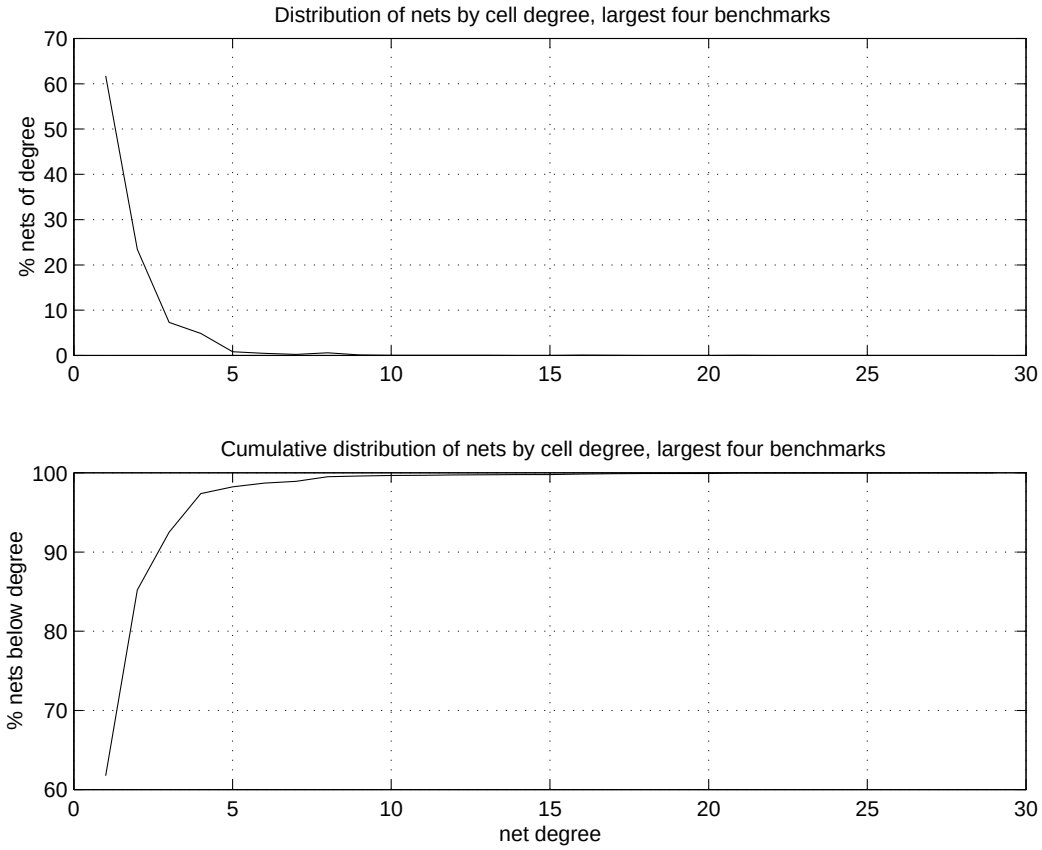

Figure 3.3: Net dize distribution for four largest benchmark circuits. The vast majority of nets have a degree below 5, with over 99% of nets having a degree below 15.

Gain evaluation in line 21 of the pseudo-code is discussed in detail in section 3.3.7.

3.3.5 Move Selection

The UTILITY heuristic focuses on the 1-opt (cell moves) and 2-opt (cell swaps) movement of cells. Other algorithms have been proposed using other movements. K-opt (group placement) moves have been used in some successful improvers [DolJoh91, KinBan89], the were shown in [KinBan89] to be too slow for simultaneous placement in a random pattern. Cell orientation (cell flipping) movements have proved to be effective (e.g.[Are95]), but are not appropriate to the UTILITY method due to the very small gains involved.

```
1.    Calculate HPWL lower bounds, and initialize net and cell utilities.
2.    for each pass
3.      sort list of cells by utility → utility list
4.      for search depth of utility list
5.        source cell ← head of utility list (worst utility), remove from list
6.        create a local search window
7.        select a target location randomly within window
8.        determine movement type
9.        if no move or swap can be made
10.         exit current iteration
11.       else
12.         save current placement
13.       end if
14.       if a move can be made
15.         move source cell to target location
16.       else if swap can be made
17.         determine target cell closest to target position
18.         swap source and target cells
19.       end if
20.       correct cell overlaps and row gaps
21.       determine gain of movement
22.       if gain > 0
23.         accept movement
24.       else
25.         reject movement and restore previous placement
26.       end if
27.     end for
28.     update net and cell utilities
29.   end for
```

Figure 3.4: Utility-based Iterative Improver Heuristic

In UTILITY, the type of movement attempted is determined by the row lengths resulting from moving or swapping cells. First, if moving a cell to a row A does not cause the new length of row A to exceed a *maximum-row-length* parameter, then the a move is attempted. Otherwise, if swapping two cells does not exceed the maximum-row-length for either row, a swap is attempted. If this also fails, the iteration fails and the heuristic continues by selecting a new source cell.

In the UTILITY heuristic, the maximum-row-length is equal to the average row length of the initial placement plus the smaller of 1% of the average row length or the average cell width. Since the initial average row length is equal to the ideal row length in a perfectly compact placement,

this ensures a small final placement area, while not greatly restricting cell movement. This is the same method used in the TimberWolfSC algorithm.

3.3.6 Local Search Window

Limiting the scope of moves within the locality of the source position has been shown to give superior results compared to unrestricted moves, when starting from a good *global* initial placement [Ken97]. This follows from the observation that a global placer places modules in a globally good, although locally suboptimal, position. After legalization, most cells should be within the locality of their final positions. By limiting the range of possible moves, the likelihood that these final locations will be found, and speed of convergence to the corresponding local minima, are increased.

Note that if we start from a poor initial position, the quality of the final solution could be drastically reduced, since a comparatively large number of cells will be far from their ideal locations. The use of a range limiter in this case will prevent those cells from finding their ideal locations. If, on the other hand, the global placer is good, there will be few of these cases, and the final quality will not be affected greatly by the use of a range limiter.

Another thing to consider is the relationship between the range limiter window, and the size of the circuit. One might be tempted to generalize that a larger problem means that moves require a larger window to avoid overly limiting the search space. However, ideally, the difference between the legalized position and the final position is a function of the legalization method used, and not the size of the problem, so a fixed initial search window size could be used on any benchmark [SecLee87].

The TimberWolf placer in [SunSec95] used a fixed window size for all temperatures in its simulated annealing schedule. This simplification made the determination of a suitable annealing schedule easier, and resulted in very good placement results. However, the tile-based improver of Kennings [Ken97] showed that altering the window size over the improvement time can increase the acceptance rate, resulting in high quality solutions in a shorter period.

Attempts to determine an adaptive window size based on circuit statistics failed. UTILITY

uses a user-selectable local search window size, so that parameters can be tuned for each individual circuit. However, on searches with many passes, an initial window size of 10 average cell widths and 3 rows either side of the source cell, and a final window size of 2 average cell widths by 1 row gave reasonably good results for all circuits, while reducing the execution time substantially.

3.3.7 Gain Evaluation

Calculating the gain of a placement is a time-consuming task. In modern search heuristics, the vast majority of time during a search is spent calculating the gains of perturbations [SunSec95, DolJoh91, Ken97].

The gain of a search iteration is the total *improvement* in HPWL due to the movement of cells. A *positive* gain is equal to the *drop* in total wirelength, and vice versa. The trivial way of calculating this would be to calculate the overall circuit wirelength before and after the move, and subtract the difference. Indeed this is the most accurate way to do it. It is also the most time-consuming way to do it.

In a cell move where no overlap is created (and therefore no other cells are shifted), only those nets connected to the moved cell are affected in any way. By examining each net affected by a move and calculating their wirelengths before and after the move was performed, the change in wirelength caused by the move can be determined. The difference in these wirelengths will be the exact gain of the movement. Calculating gain in this manner is the only way of calculating wirelength gain exactly, but it is the single most time-consuming step in iterative improvement. Any improvement to gain calculation can result in dramatic improvements in running time [SunSec95].

Ideally, the fewest nets that must be examined are only those connected to cells being moved. In standard cell placement where cells may have many different widths, this situation only arises when swapping two equally-sized cells, a and b, where only nets $N_a \cup N_b$ must be examined (nets connected to either a or b).

Swapping equally-sized cells does not occur frequently during improvement. It is much more common that either a single cell is moved, or unequally-sized cells are swapped. In both of these cases, overlap is generated when one cell is moved to a location that is already partially

occupied by other cells. To achieve a legal placement, these "occupying" cells must be shifted to non-overlapping positions in the row to make room for the moved cell.

Unlike in the ideal situation described, these shifting movements alter not only the wirelengths of nets connected to the cell being moved, but also the wirelengths of all nets connected to the cells shifted out of the way as well. Whereas in the ideal case only a few nets might be affected, when overlap is created a large number of nets are affected. In a large benchmark, potentially tens of thousands of nets can be affected by the movement of a single cell! It is in these situations that calculating movement gains precisely can result in huge run-times.

In [SunSec95], Sun examined a method to reduce the number of cells to be shifted, by allowing gaps to exist in the placement. If the rows are not compacted after a movement is made, then the source row of a move, or the row containing the larger cell in a swap do not need to be shifted. This can significantly reduce the number of nets that must be updated.

Usually, when cells are shifted, it is always in a single direction, depending on how the row data structures are implemented. Another technique to reduce the number of affected nets proposed in [SunSec95] is to select the direction of a shift in different directions depending on the insertion point. A simple technique used to maintain a compact placement is to keep the left edge of a placement flush with the core edge, and have the right edge floating with the row length variations. To maintain this structure, all shifts are towards the right. This means that, on average, half of the cells are shifted for each move. Sun and Sechen proposed that by abandoning this flush-left structure and allowing rows to be shifted to the side of the row closest to the inserted cell, even fewer numbers of cells need to be shifted. The problem with this approach unmodified is that the placement tends to "spread out" along the row, with *row gaps* concentrated in the center of the row. Movement activity dies quickly, as more and more move attempts are rejected due to row length constraints. Sun and Sechen did not explain how they overcame this problem.

The approach used in UTILITY is to allow row gaps, and to use a new method for selecting the direction of shifting cells. Instead of shifting to the closest side of the row, UTILITY shifts towards the *closest gap* large enough to accommodate the overlap introduced by the inserted cell. As can be seen in Figure 3.5, by inserting cell A into the first row and shifting cells to the closest

side of the row (left), the final row length, even after absorbing some overlap by gap a, will exceed the core, which is highly undesirable. By shifting to the right, the row gap b can absorb the overlap without exceeding the core dimensions. Another feature of this method is that it finds the fewest cells to be shifted if a maximum row length is to be maintained. For example, if cell A is moved to the second row of Figure 3.5, the row gap c is large enough to accommodate the overlap of cell A, but shifting right will actually shift fewer cells. The combined size of gaps d and e can also absorb the overlap, but the furthest of the two, e, is still closer than c. By searching for the closest gap from the target location, this mechanism ensures that a small number of cells is shifted (and therefore a small set of affected nets), while maintaining a stable core size.

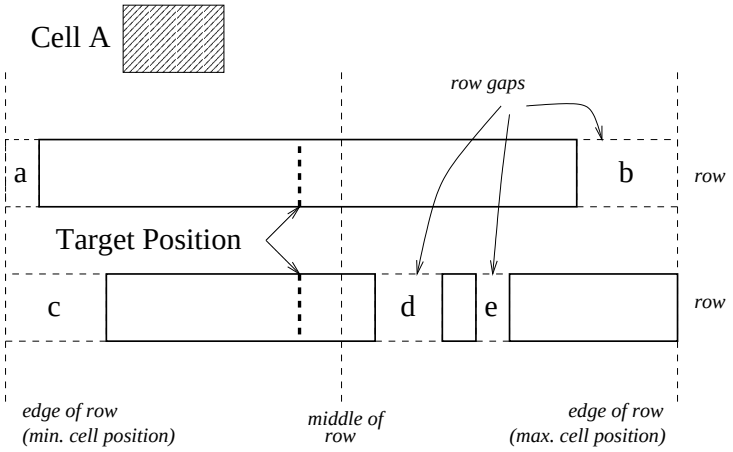

Figure 3.5: Determining cell shift direction

In UTILITY, we use the same gain estimation mechanism as [SunSec95], which reduces the time of calculating the gain of a movement by approximating the affect of shifted cells on wirelength changes. In this gain model of shifting cells, the change in wirelength for the cells being moved or swapped is calculated exactly, but the change in wirelength for the cells that must be shifted to eliminate overlap is *estimated*, as demonstrated by the three-cell net in Figure 3.6. The half-perimeter wirelength of a net is due to the placement of pins at the perimeter of a net's bounding box (cells A and B in Figure 3.6). Moving a pin on this perimeter, such as cell A, will generally alter the HPWL by an amount equal to the pin's displacement, for small moves.

Moving a pin that is not on the perimeter, such as that of cell C, will not alter the HPWL for small moves. The value of these observations is that, rather than calculating the bounding box for the net after each cell's shifting by examining all pins on the net, the change in wirelength can be approximated by simply adding or subtracting the shift amount from the previous net wirelength as appropriate. Since hundreds or thousands of pins do not to be repeatedly examined, this method can dramatically reduce execution time of the search[SunSec95].

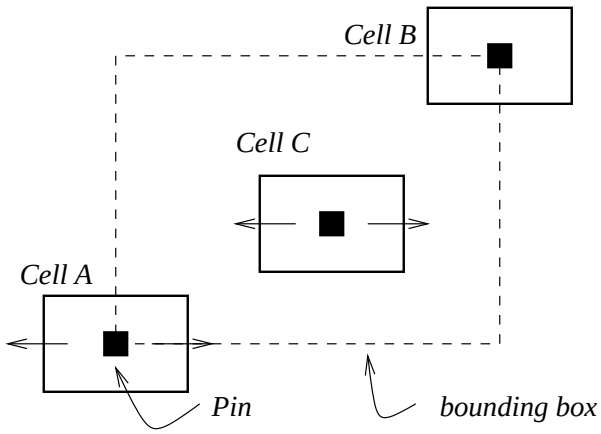

Figure 3.6: Estimating change in HPWL for small cell displacements

The shifting model for gain estimation is not perfect. For larger moves, such as when a pin not on the perimeter is moved outside of the perimeter, this estimation model will not hold since it will not indicate the true increase in HPWL. Another case not reflected accurately is when a cell is moved such that it is no longer on the perimeter. For example, if cell A is moved to the right a small amount, the HPWL is decreased by the amount of the move, but if it continues to move right until it passes cell C, the actual HPWL will not continue to decreases since the pin on cell C becomes the new perimeter of the net's bounding box, but the shifting model will not reflect this fact.

Despite these inaccuracies, since the maximum displacement of moves in the UTILITY heuristic is restricted by the local search window, most displacements are small compared to the size of the bounding box, so the shifting model approximation is fairly accurate in practice.

To further minimize the adverse effects of large nets in a circuit, large nets are ignored during move gain estimations. The reasons are the same as in section 3.3.3, namely that estimating gains for large nets takes requires extra time without significantly altering the placement outcome. Sun and Sechen in [SecLee87] showed that, when used with the shifting gain estimation model, limiting gain estimation to nets smaller than 30 cells actually *increases* estimation accuracy slightly, and reduces run time by over 80% by reducing the time required for computing movement gains. Therefore, only nets smaller than 30 cells are considered during gain estimation in this chapter.

3.4 Results

Numerical results were obtained using the full set of MCNC benchmark circuits, introduced in section 2.5.

3.4.1 UTILITY as a Primary Improver

The UTILITY heuristic was developed to be a fast, greedy heuristic, exploiting knowledge of the current placement to select moves with a good probability to improve the placement quality. The utility list search depth of a UTILITY search determines the size of the search space that can be examined during a pass. A smaller search depth will concentrate on fewer possible moves, and therefore limits the size of the search space. However, a smaller search depth will also not waste time on well-placed cells that are very unlikely to be improved by a move.

Figure 3.7 illustrates the behavior of the UTILITY heuristic with varying utility list search depths, using the Industry2 large benchmark as an example. In the legend, 20% means that the lowest 20% of the utility list is examined in a pass, before the utilities are re-calculated for the next pass. Here, UTILITY was used as a primary improvement heuristic, meaning that was used alone to achieve the final solution quality, starting at a good initial solution. However, UTILITY was not designed to be used as a primary improvement heuristic. Figure 3.7 also compares the UTILITY heuristic to a good greedy random-move-selection iterative improver to illustrate this. The improver used was the TILE heuristic developed by Kennings in [Ken97], and has been shown

to give very competitive final solutions. We used the ARP global placer developed by Etawil in [EtaVan98] to generate initial solutions. For both heuristics, the same-sized local search window was used.

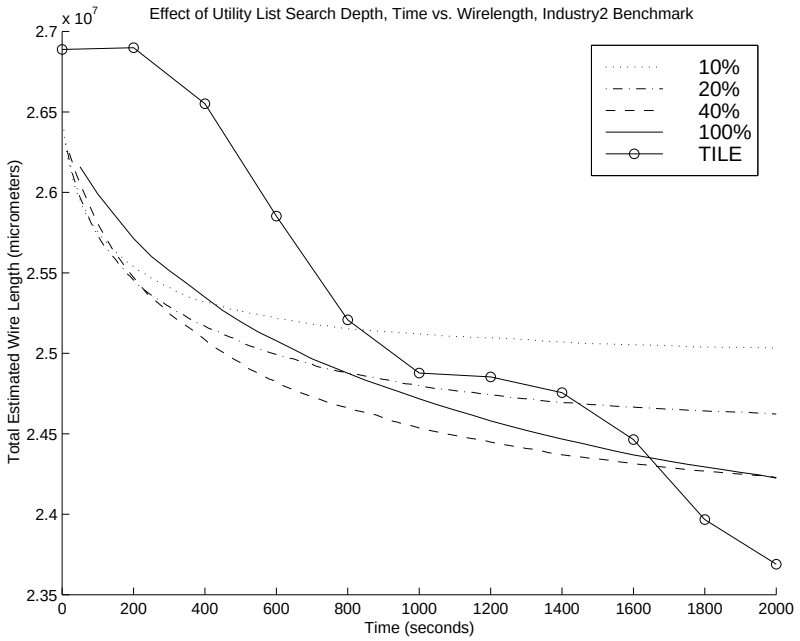

Figure 3.7: Comparison of UTILITY to TILE as a primary improver, showing affect of different utility list search depths

The general behavior of a utility-based is demonstrated. By using a utility list with a fixed search depth, a particular region of the search space is examined rigorously. For a small search depth, this means that the very worst-placed cells are emphasized, and a large amount of improvement is seen in a very short time. However, this traps the solution in a local minimum very quickly. Examining the results for the 10% search depth, we can see that this is true. The solution converges very quickly, but the quality is poor. Increasing the search depth increases the number of cells that are examined, increasing the search space, so convergence takes longer, but final solution quality is greater. Search depth can be traded-off with convergence speed.

The TILE improver, which is a greedy method which rapidly moves large numbers of cells, examines much more of the search space than UTILITY. Starting from a "good" solution, there

is still much possible improvement to be made, which is exploited efficiently by TILE. Even at the slowest convergence rate (when the entire utility list is examined), UTILITY converges at an inferior solution to that of TILE.

3.4.2 UTILITY as a Post-Processing Improver

UTILITY can function as a primary improvement method, but it was not designed to be so; UTILITY was designed as a fast greedy search heuristic. Because of this, it is better suited as a "post-processing" improver, whereby it is used to quickly achieve a slight improvement in already-high-quality solution. Figure 3.8 shows the performance of the utility heuristic on a placement *after* being improved by ARP and TILE. In this case, the initial solution is of very high quality, meaning that the solution is very close to a local minimum. Because of this, there are few moves remaining that will improve the solution. The initial solution was obtained by running the TILE improver until it converged to a good solution. After the TILE had finished, the UTILITY heuristic was run for 3000 seconds on the industry2 benchmark, varying the utility list search depth.

As a post-processing step, the UTILITY displays the same overall behavior as when used as a primary improvement method. The same generalization holds true: decreasing the utility list search depth decreases final solution quality but increases convergence rate. This is the characteristic that makes UTILITY poor as a primary improver, but good as a post-processing improver.

Comparing Figures 3.7 and 3.8, it can clearly be seen that the rate of convergence of UTILITY has drastically diminished after the TILE improver has been used. The reason for this is simple: the starting solution in Figure 3.8 is much better than that of Figure 3.7, and so there are far fewer moves that can improve the solution.

As a primary improver, the convergence rate of UTILITY is so fast for small search depths compared to TILE that it is not practical to sacrifice solution quality for convergence speed, since increasing the convergence speed by a comparatively small time can greatly increase solution quality. Since many moves can result in improved placement cost, it is not practical to waste time

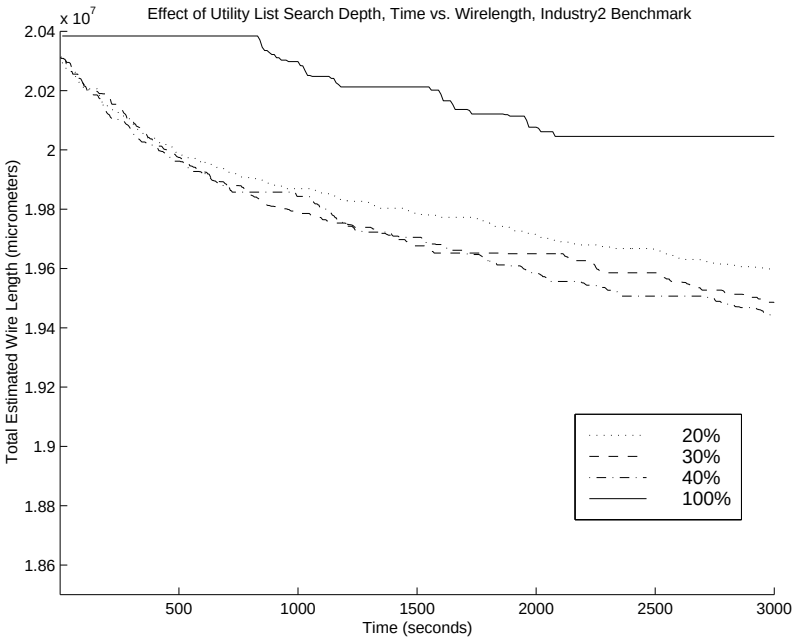

Figure 3.8: UTILITY effect of varying search depth on total wirelength, with a constant window size of 5x2

with utility overhead when a simple random-move selection will achieve good quality solutions in a small amount of time.

After the placement has been improved to a high-quality solution, however, a random-move selecting heuristic, such as TILE, spends almost all search time examining moves that do not improve the solution. UTILITY, on the other hand, focuses on the worst-placed cells, and therefore greatly increases the proportion of moves that increase the solution quality. The same basic behavior still holds true: faster convergence results in lower final quality, but because convergence is so slow, it is much more reasonable to accept an inferior solution to save time.

Table 3.1 summarizes the typical performance of the UTILITY search heuristic when used as a post-processing improver. Results labeled "No Util" were attained by running ARP followed by running TILE until it converged. Results labeled "Util" were attained by performing 100 iterations of UTILITY, on the results attained in "No Util". UTILITY was run searching 20% of the utility list with a window size of 5x2..

circuit	No Util *WL*	Util *WL*	Imp. %	No Util *Time*	Util *Time*	Incr. %
Fract	0.035	0.035	0	9.2	10	8.7
Primary1	0.84	0.84	0	127	131	3.0
Struct	0.44	0.44	0	186	197	5.5
Sm. Avg.	-	-	0	-	-	5.7
Industry1	1.70	1.68	1.2	602	617	2.4
Primary2	4.26	4.22	0.9	611	628	2.8
Biomed	2.29	2.21	3.5	1327	1364	2.7
Me. Avg.	-	-	1.9	-	-	2.6
Industry2	20.2	20.1	0.5	3344	3411	2.0
Industry3	54.1	54.0	0.2	4400	4475	1.7
Avq.small	10.2	9.97	2.3	6853	6963	1.6
Avq.large	11.4	11.2	1.8	7735	7857	1.6
Lg. Avg.	-	-	1.6	-	-	2.3
Average	-	-	1.1	-	-	3.4

Table 3.1: UTILITY performance as a post-processing improver.

The results in Table 3.1 show that a small amount of improvement can very quickly be achieved when UTILITY is used on a placement that is already of very high quality. For the large benchmark circuits, an average improvement of 1.6% was achieved with an increase in total execution time of only 3.4%. Therefore, while UTILITY was not very effective as a primary improver, it is quite effective as a post-processing improver.

3.4.3 Local Search Window Size

The size of the local search window affects the size of the search space that can be examined from any particular placement. Figure 3.9 shows the affect of varying the local search window size of a UTILITY search, with a fixed search depth of lowest 20% of the utility list.

3.4.4 Cell Utility Net Size Cut-off

In section 3.3.3, it was proposed that large nets should not be used in cell utility calculations. The reason for this was that, because large nets are very difficult to improve, their contributions of net utility could be misleading to cell utility values, and this property should be reflected in

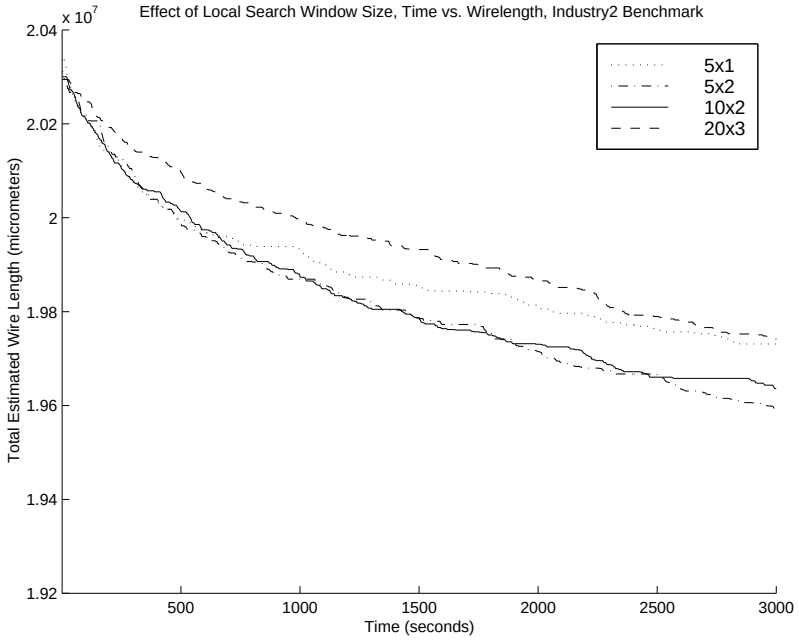

Figure 3.9: UTILITY effect of varying window size on total wirelength, with a constant search depth of 20%

the cell utility calculation. Figure 3.10 summarizes the effect of different net size cut-off values to final wirelength, using the Industry2 benchmark as an example. Results were attained using the ARP global placer, the TILE improver, and then 100 passes of the UTILITY search improver.

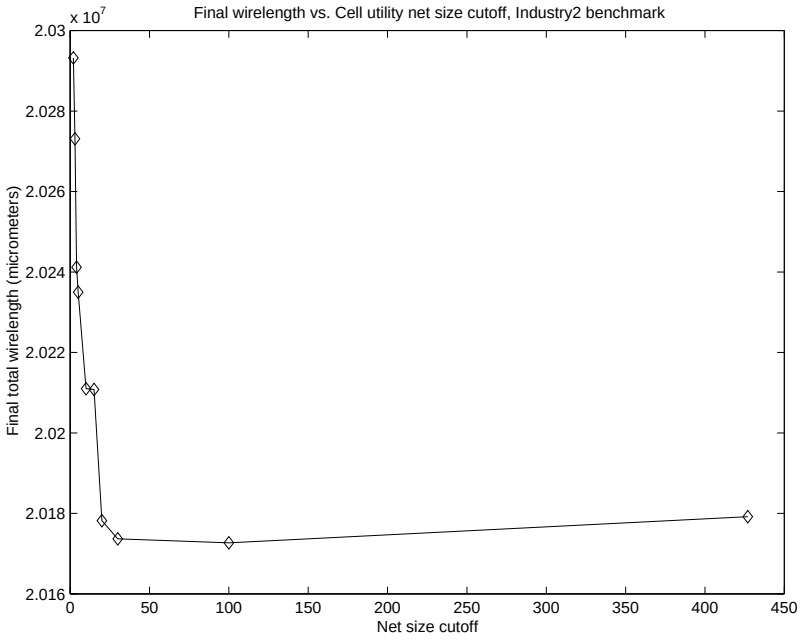

Figure 3.10: Effect of net size cut-off in cell utility calculation, showing how a cut-off below 30 cells results in significantly poorer solutions. Results are for the industry2 benchmark with a maximum net degree of 427.

Figure 3.10 corroborates the assumptions made in section 3.3.3. In agreement with the conclusions presented in [SecLee87], disregarding large nets does not adversely affect solution quality, and actually provides a slight improvement over a solution considering all nets.

3.4.5 Effectiveness of Utility as a Measure of Cell Placement Quality

In this chapter, utility theory was used to describe the quality of placement of a cell, in order to determine good candidates for search moves. Figure 3.11 shows that there is a high correlation between improvement in cell utility and improvement in wirelength, especially for the largest benchmarks.

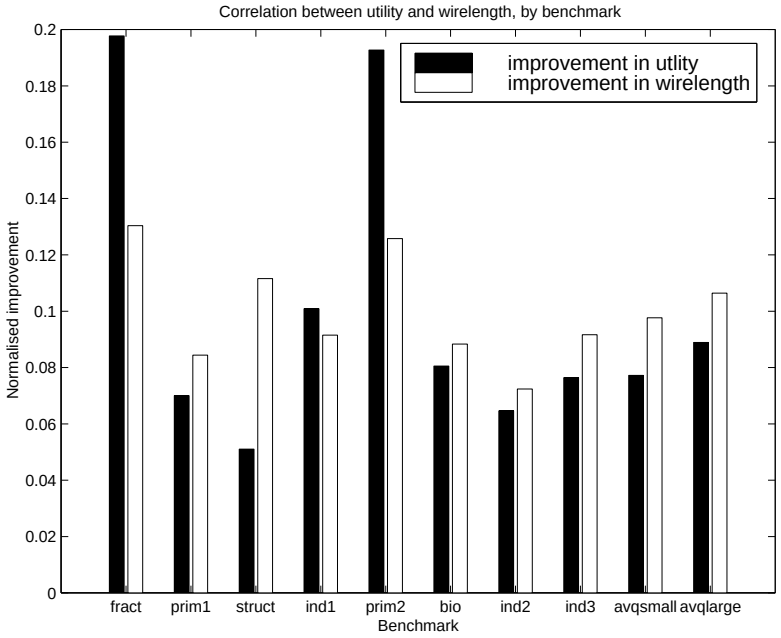

Figure 3.11: Correlation between wirelength improvement and cell utility, using UTILITY heuristic. Normalised improvement is the ratio of final value to initial value, normalized to the initial value.

Figure 3.12 shows the distribution of total half-perimeter wirelength by net size, and shows that the vast majority of total estimated wirelength is distributed in small nets. This is consistent with the observation in section 3.3.3 that most nets are between 2 and 5 cells in size.

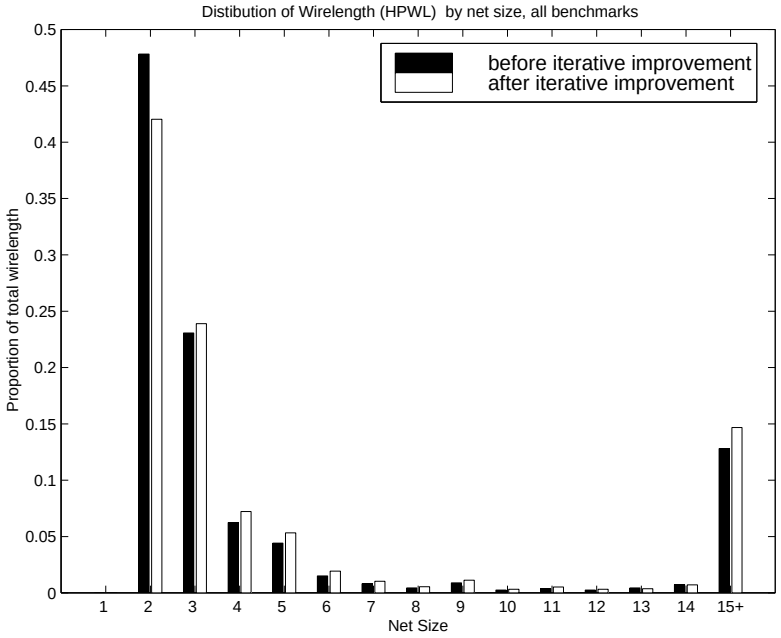

Figure 3.12: Distribution of total half-perimeter wirelength by net size. Porportion of total wirelength is the total wirelength of all nets for each net size, divided by the total wirelength of all nets.

Figure 3.13 shows that after improvement with UTILITY, the net utility of small nets between 2 and 5 cells in size is increased. For two-cell nets, net utility is increased by a factor of almost 100%! This indicates that, in terms of the utility measure, net lengths are closer to optimum.

Since there is a high correlation between improvement in wirelength and improvement in cell utility, our measure of cell utility is a good measure of a cell's placement's closeness to optimality.

3.5 Summary

In this chapter we examined a novel new approach to overcome the problem of random move selection in a deterministic (greedy) search heuristic. By measuring the *utility* of a cell's placement,

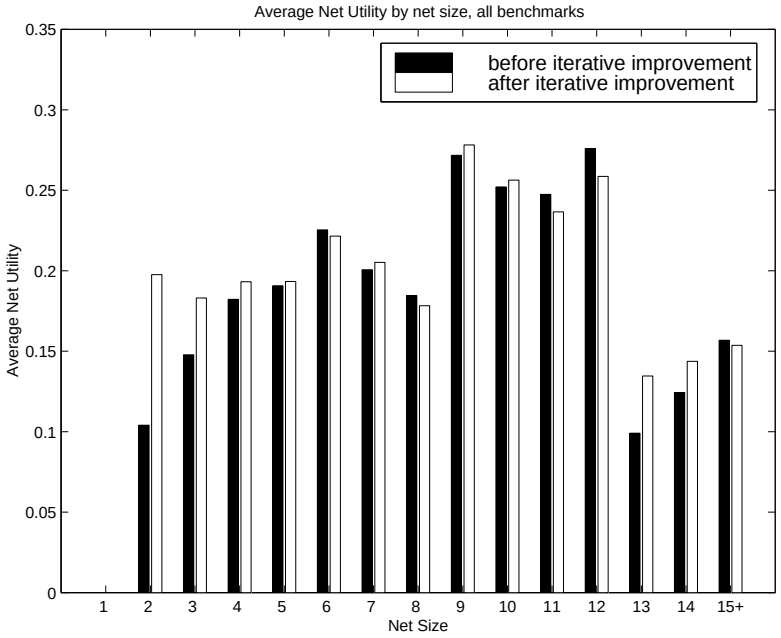

Figure 3.13: Average net utility by net size.

moves are chosen that are statistically more likely to improve than if the moves were chosen randomly. By concentrating computational effort on cells with worse-than-average placements, greater gain can be achieved with less effort than that which a randomized method could achieve in the same time. In this way, it is possible to reduce the computational impact of examining moves that will be rejected.

It was shown that using utility as a measure of placement quality was effective. Improving the cell utility of poorly-placed cells improved the total wirelength of the solution, and solution convergence time was dramatically increased with smaller utility list search depths, so that "good" solutions were found quickly. However, when compared to a good deterministic search heuristic as a primary improvement heuristic, the results using utility were not competitive. The convergence time for the utility-based heuristic was too short to adequately examine the search space, and so the solutions were of very low-quality.

Utility-based improvement was shown to be best suited for use as a fast post-processing im-

provement step, after significant improvement has already been made by another improvement method. Most good improvement methods, such as simulated annealing, achieve a large degree of improvement by using stochastic methods in initial phases of improvement, in which a large number of non-improving moves are accepted in order to escape local minima. Utility-based improvement, by contrast, is exactly the opposite of this. Random move selection is avoided, and gain is accepted deterministically. Because of this, utility-based improvement cannot function well as a primary improvement heuristic.

However, when the placement quality is already high, a deterministic approach is desirable. Again using simulated annealing as an example, in late placement phases, when the solution quality is getting very good, simulated annealing behaves very deterministically, accepting very few non-improving moves. If more non-improving moves are accepted, the solution will not converge to a local minimum, and at worst, the solution will find a new, poorer-quality local minimum, defeating the purpose of using hill-climbing. Because utility-based improvement is highly deterministic, quickly finding a local minimum, it is well suited to achieve any *post-processing* improvement.

In the next chapter we investigate another way to handle highly complex problems in ASIC design. In this chapter, the complexity of the standard-cell placement problem was handled by reducing the size of the search space by using utility to focus search efforts on poorly-placed cells. In the next chapter, *circuit clustering* is introduced which actually *reduces* the search space of the problem, rather than trying to confine a search. The circuit size is reduced, so that the complexity of the improvement task itself is reduced. This method also presents an interesting challenge to restoring circuit complexity, in which utility-based improvement plays an important role.

Chapter 4

Clustering-based Placement

4.1 Introduction

4.1.1 Motivation

As mentioned previously, the size of standard-cell placement problems is increasing at a substantial rate. Even the outdated benchmarks available to the academic world are very large by combinatorial scales. Computation speed increases over time as well, but because of the complexity of the placement problem, a doubling of the circuit size cannot be handled in the same time by a doubling of the computation speed. A placement heuristic that produces excellent results, but takes weeks or months to run is often useless to the modern just-in-time fabrication mentality. In many cases, even a heuristic that runs in days is considered too slow, when a few years ago it was considered commonplace.

This demonstrates the need for good but fast placement heuristics. Commonly-used heuristics that were appropriate for smaller circuits can not stand up to the demands placed on them by larger circuits, because the run-time complexity of these heuristics is simply too large. The need for pseudo-linear time heuristics is evident (pseudo-linear meaning very low-order polynomial, such as $O(nlogn)$ and preferably less than quadratic ($O(n^2)$)).

There are two techniques currently in use to deal with this problem. The most obvious method

is to implement faster heuristics at the cost of lower-quality solutions. The other is to attempt to reduce, or "cluster", the size of the circuit into a less-complex form.

4.1.2 Circuit Coarsening

The first approach to a placement problem is to solve it in a top-down fashion, by considering globally the best positions for cells in a placement. This is, of course, incredibly time-consuming, and simply isn't practical for large problems without relaxing the problem.

The more conventional approach is to use a bottom-up, iterative improvement approach, which attempts to find a good overall solution by looking at one or a few cells' movement at a time. This is less time consuming, but also less beneficial since only a small local improvement is considered at a time.

Techniques have been attempted to "blend" the two approaches to get the best of both worlds. One example is simulated annealing, by definition an iterative improver and so a bottom-up method, but at its high temperature regime, it is know to behave in a top-down manner [SunSec95]. For this reason, simulated annealing is often referred to as a "pseudo-top-down" method, since over the course of its improvement it considers the global placement of each cell. It has been mathematically proven that the simulated annealing method can give a global optimal solution given a long enough time [MitRom86].

A more recent approach to combining these techniques is called hierarchal improvement, and is a two-step procedure, first proceeding bottom-up, and then top-down. The bottom-up technique is clustering, and involves the grouping of highly connected cells into clusters and clusters into larger clusters, while the goal of the top-down method is to determine the location for all the clusters, and then the location of all cells within those clusters [MalGro89]. The goal of this is to reduce the number of entities that need to be improved, and the number of interconnections between them, through the bottom-up stage. This reduces the search space by reducing the degrees of freedom for cell moves, making a top-down method more feasible.

This approach was first applied to the linear placement problem in 1972 with Schuler and Ulrich's paper [SchUlr72] and has since been applied heavily to the partitioning problem. Only

recently has it been applied to the standard-cell placement problem, and then only in limited usage [SunSec95, MalGro89]).

Solution quality doesn't necessarily need to deteriorate to achieve computational gains, since search efficiency is also improved. This conclusion follows by observing that without clustering, for two highly connected cells to move together, first one would have to be selected to move, then the second would have to somehow find its way close to the location of the first one. This would take much longer on a non-coarsened circuit because several moves would have to be tried before they can come together [MalGro89] (This of course assumes that the clustering method is good, as any poor clusters made at the bottom-level step will be "fixed" at the high level).

4.1.3 Clustering v. Coarsening

The terms "clustering" and "coarsening" are encountered frequently in the literature, and the terms can cause confusion. Clustering usually refers to generating a small number of super-cells, each containing a large number of cells. This is analogous to a k-way partitioning where k is reasonably large. Coarsening usually refers to generating a large number of super-cells, where each super-cell consists of only a few, or even just one, cell. The distinction between the two is "fuzzy". In practical use, the terms are generally interchangeable. We will endeavor to refer to a group of cells as a "cluster", and the act of creating clusters as "coarsening" (since it is somewhat more accurate).

4.1.4 Measurement of clustering quality

Despite active attempts to find one, there is presently no known metric for directly measuring the quality of a clustering [Alp94]. Attempts have been made to quantify and define metrics, but the ones found are specific to a single sub-class of clustering heuristics (the random-walk method described later), and are based on clustering-based-partitioning results.

The only agreed-upon measure of a clustering's quality is by the final improvement obtained using the method. In other words, we must infer the quality of our clustering method from the quality of our whole placement heuristic. This is obviously not an ideal situation, since final

solution quality is a factor of many variables, including the size and statistics of the circuit being placed, the placement heuristics used, and the length of improvement time allowable. The best that can be done to distinguish clustering methods apart is to attempt to isolate the effects of the clustering method from the improvement methods by testing in different situations with different improvers and different benchmarks. As we discuss later, we have attempted to test this solution space as thoroughly as time reasonably permits.

4.1.5 Goals of Clustering and De-clustering

As we mentioned above, the quality of the clustering method can only really be measured by the quality of the final solution. We only know the final solution after the clusterer has done its job, so what objective does our clusterer use?

The principle behind coarsening a circuit is to: (a) reduce the problem size, while (b) keeping the quality of the solution as good as for the flat circuit [Kar97, MalGro89]. Obviously, the computational time of the coarsening method should also be kept small. Therefore, a good clustering method should quickly identify groups of cells which will eventually end up together in the final placement stage [SunSec95].

In doing this, there have been some common goals *empirically* give consistently good results, both for the partitioning problem and for the placement problem. First, the physical size of clusters should vary over a small range [SunSec95]. The reason for this is that if the range is large, gain estimation errors get larger, and the size difference inhibits cell movements in the iterative improvement. Second, the size and number of nets should be reduced [Kar97]. This is a direct consequence of our primary goal of reducing the problem size, but is frequently overlooked in the literature, where the concentration is on reducing the number of cells. The reason for this is because placement methods perform better, and cost estimation is faster and more accurate, with fewer and smaller nets. Nevertheless, there is general agreement that reducing the number of nets, and choosing clusters to reduce the most nets, is preferable, and gives best results. This implies that highly connected cells have been clustered since many nets will become localized (connected only to cells within the same cluster) [SunSec95, Kar97, MalGro89].

Therefore, the highest priority in a clustering method should be to localize, or increase the potential to localize, nets to a cluster. It should also ensure that the cluster sizes are not excessively diverse, and that the size of nets is reduced if not entirely eliminated.

The goals of de-clustering are comparatively simple. First and foremost, the original flat circuit should be obtained after de-clustering, i.e., the original circuit should not be modified. Second, the quality of the solution at the flattened level should be as close as possible to that of the clustered levels. The measure of this quality is simply the increase in total wirelength while descending the hierarchy towards the flat circuit.

4.1.6 Contributions

In this chapter, we propose a technique to reduce the problem size, while keeping the execution time to a minimum. We reduce the problem size using a circuit coarsening technique, and we reduce the complexity of the search by examining pseudo-linear time heuristics for performing the clustering and de-clustering. We consider some fast heuristics that have been applied successfully to the circuit partitioning problem, and present several modifications to consider the particular demands of the placement problem. We also apply a fast legalization heuristic to reduce the de-clustering penalty.

4.2 Previous Work

This chapter investigates the application of circuit clustering to the placement problem. Circuit clustering has been a revolutionary development to design automation. By reducing the complexity of the problem, very large industrial problems can be dealt with in reasonable amounts of time. This has been the most significant benefit of circuit clustering - enabling heuristics to operate on very large problems. Circuit partitioning has been the major area for previous clustering research, with literally hundreds of papers published in conferences and journals in recent years. Yet very little academic work has been done for the clustering-based placement problem, a much more complex problem than circuit partitioning. Less than half a dozen papers have been published in

the last decade on clustering-based placement, and only a single paper has addressed multi-level placement [SunSec95]. The major purpose of this work is to attempt to apply what has happened in the partitioning world to the placement problem, and to note any relevant issues that should be considered that are particular to placement.

In the rest of this section we first provide a brief survey of the major successful clustering methods that have been applied to circuit clustering, and then present the previous work that has been done in clustering-based standard-cell placement.

4.2.1 Partitioning-based Clustering methods

The partitioning problem (see section 2.2.1) has long been a "test-bed" for experimenting with advanced search heuristics, due to the wide uses of partitioning in various fields such as network routing, computer memory management, and, of course, physical design automation. As a result of this work, a staggering variety of approaches to partitioning have been applied, and many perform very well.

As previously mentioned, one can view the problem of clustering a circuit as the equivalent problem of partitioning a circuit into a very large number of small partitions. This may seem odd - to solve a partitioning problem in order to better solve another partitioning problem - but the structure of the problems is actually very different. In general application, the final solution to the partitioning of a circuit should be very high-quality. In contrast, partitioning-based clustering does not need to be so accurate, since many approximation are made in the clustering heuristic itself.

This observation has given new life to many primitive clustering methods that were originally designed with speed in mind, but have given way to more advanced, but much slower, methods which give high-quality solutions. Some of these methods are ideally suited for clustering, since speed is one of the primary considerations when forming clusters.

Ratio-cut circuit partitioning

This method finds natural clusters by minimizing the interconnections between clusters [YehChe92]. It generally finds few clusters with wide range of sizes, and the number of clusters is not known in advance. Also, most heuristics have a quadratic or greater time complexity, making them unsuitable for a fast coarsening technique [YehChe92].

Random Walk

This is another partitioning based-method that first finds a random non-repeating sequence of cells by "walking" from one cell to another by their interconnections [ConHag91, KahKan92]. The "steps" are usually selected at random, but can consider other attributes as well, such as the weight or number of interconnections. Clusters are then formed by performing a linear partitioning on the sequence. The length of the random sequence is quadratic, and total complexity of the heuristic is cubic. Also, the walk and partitioning heuristic can not easily consider the cell sizes.

Eigenvector/Spectral methods

Eigenvector methods are attractive in circuit partitioning because they are fast and require low storage overhead (e.g., [HagKah92, AlpKah96]). First, the net list adjacency matrix A is created, which requires the circuit hypergraph to be modeled as a graph. The diagonal matrix $D = (d_{ij})$, with $d_{ii} = \sum_{j=1}^{n} a_{ij}$, is used to form the Laplacian matrix of A, given by $Q = D - A$. The first d eigenvectors of Q are found, where d is a user-specified parameter. Then, the eigenvectors are combined, either through summation, binary encoding, or some other way. Modules with similar-valued entries in the combined eigenvectors are assigned to the same cluster.

This is a highly mathematical approach to the clustering problem, which can best be viewed as follows: Any eigenvector is a one-dimensional (i.e., linear) placement of the modules in a net list. Closely-valued entries in the eigenvector indicate strong connectivity between the corresponding modules. If several eigenvectors (i.e., several one-dimensional placements) are obtained and combined in some meaningful way, highly connected regions of the net list can be discerned.

Eigenvector-based methods can be implemented very quickly, limited only by the eigenvector

calculations which take an expected $O(n^{1.4})$ time using the Lanczos iteration [AlpKah96]. The main disadvantage of these methods is that they view all modules as being equal sized, with no straightforward way to weight individual modules.

4.2.2 Constructive Clustering methods

Partitioning-based methods rarely consider the sizes of clusters as an objective - a consideration of major importance to the placement problem - and partitioning-based methods that do consider sizes tend to be prohibitively time-consuming (e.g., [YehChe92]).

Another approach to the clustering problem the constructive method, in which clusters are "grown" from selected *seed* cells. This is similar to the constructive placement approach, discussed in section 2.4.3. There has also been previous work in the partitioning field on constructive partitioning, but results tend to be sub-standard to partitioning-based methods when compared on solution quality obtained, and are often used merely to generate a better-than-random initial solution for an iterative improvement technique [Ken97]. However, these methods tend to be flexible to size constraints - an important consideration in forming clusters.

Prioritized Attributes

In this constructive method, modules are merged according to an attribute list [MalGro89]. The attributes considered include terminal count, common net count, number of nets localized, common net fan-out, and cluster size. The major advantage of this method is that it can be easily adapted to consider different priorities and combinations of attributes. The main disadvantage is that the ideal priorities and attribute list is not the same for all circuits. The time complexity for this method depends on the attributes considered.

(k,l) Connectivity Method

In this method, clusters are formed using a measure of graph connectivity. Clusters are formed by joining k-connected components in a circuit into a cluster. To limit the search for connected components, only components connected by a path with length l are considered [GarPro90].

The choices of k and l are difficult to select. The method has a polynomial complexity in $O(n^{kl})$. As well, for choices of $l > 1$, unnatural results can result, since cells further apart from each other are more likely to be joined than those closer together (since there are more paths between them, as measured by the heuristic, and hence meet the k-requirement to form clusters).

hMetis

The methods proposed by Karypis et. al. [Kar97] are simple connectivity based methods, clustering edges or hyperedges in a hypergraph in linear-time. The method implemented in the hMetis software package [Kar99] greedily clustered entire nets in one pass, and then clustered remaining cells in a second pass. No constraints on cluster sizes or numbers were considered. Nevertheless, this very fast and simple method produced the best results at the time of its publishing. However, these methods have never been publicly applied to placement.

The simplest method presented is the "edge-connected clustering", or "EC" method. A cell is chosen, and a cluster is formed by pairing it with a randomly-chosen connected cell. In a graph, the result is to eliminate a graph edge, but in a hypergraph the result is merely to reduce the size of the hyperedge (unless the hyperedge only connects the two cells, of course).

Results using EC were of lower quality than the hyperedge based methods, the best of which is the "modified hyperedge clustering", or MHEC, method. MHEC functions by examining the net-list in two passes. In the first pass, each net is visited in order of increasing size, and the net is clustered if none of the cells on the net are already assigned to an existing cluster. In the second pass, again each net is visited in order of increasing size, and all remaining unclustered cells on each net are formed into a cluster.

Apart from performing well in the partitioning problem, it has many properties that make it appealing for application to the placement problem: linear time complexity, fairly even cluster sizes, and good circuit size reduction. The focus of MHEC is to reduce the number of nets in the circuit as greedily as possible. In the first pass, as many small nets are eliminated (clustered) from the circuit. The second pass has the effect of "almost" eliminating nets, in that in subsequent clustering levels it will be easier to eliminate the net. By visiting nets in order of increasing size,

this method implicitly attempts to cluster smaller nets first, so that the number of nets in the problem is more likely to be reduced.

4.2.3 Hierarchal Clustering

Early methods of clustering performed the desired circuit size reduction in a single level (e.g. [MalGro89]). Research has recently shown that clustering in steps (illustrated in Figure 4.1), reducing the circuit size gradually by adding intermediate levels to the hierarchy, produces superior results by permitting more gradual de-clustering [Kar97]. This gradual clustering is often called "multi-level" or "hierarchal" clustering.

During de-clustering in a single clustering level heuristic, the difference between positions in clustered cells and flat circuit cells can be substantial, and significant iterative improvement is necessary to achieve a high quality solution. In a multi-level heuristic, much smaller differences are created between levels of the hierarchy, because it is built slowly. During de-clustering, these differences are more easily managed by simple interchange heuristics, resulting in a superior quality solution in a shorter amount of time.

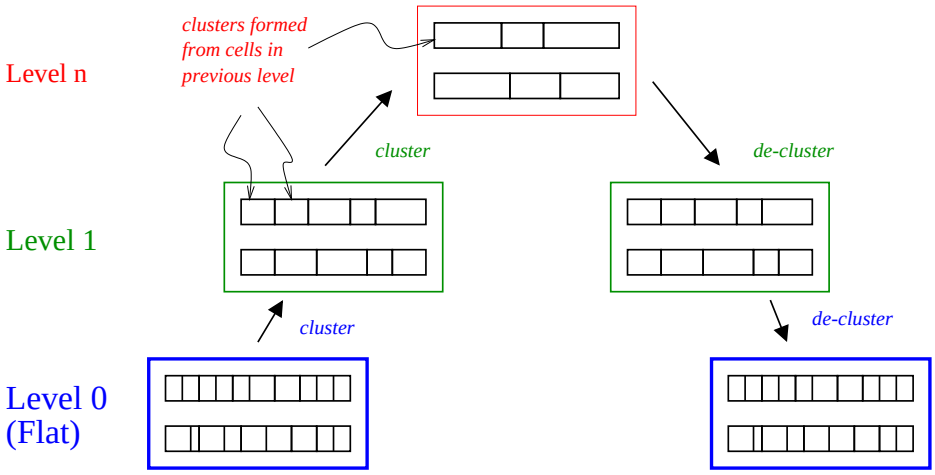

Figure 4.1: Multilevel Clustering Hierarchy: Majority of placement effort is executed at top hierarchy level, when the circuit is smallest.

4.2.4 Application to the Standard-Cell Placement Problem

As previously mentioned, very little work has been publicly done applying clustering to the standard-cell placement problem. So far only two main efforts have been put forth, and one of these has since released their product commercially, ending the release of academic information.

TimberWolfSC

With the last public release of TimberWolfSC [SunSec95], a clustering-based simulated annealing heuristic produced the highest-quality results up to that time. The clustering method used was partitioning-based, minimizing a cost function of a weighting of the pins of the cluster, through an application of simulated annealing. The method explicitly emphasized clustering nets with small fan-out (to reduce overall number of nets) and the importance of maintaining similar-sized clusters. In theory, the clustering method is linear time, but it uses simulated annealing (which is decidedly NOT fast).

The clustering method in TimberWolf also constrains maximum and minimum cluster sizes, and gives a pre-defined number of clusters, which can mean that clusters are "forced" to fit the constraints, but that is generally not a problem with huge circuits. Overall, the clustering method allowed very good results for placement, but was still fairly time-consuming.

Mallela and Grover (Prioritized Attributes)

Mallela and Grover [MalGro89] identified the need for a fast clustering method, and used the prioritized attributes method to "grow" clusters from randomly-selected seed cells based on the strength of their connections . Their method finds a sub-optimal matching solution pairing cells to seeds by iterating over many passes to find good potential clusters based on the attributes, while limiting cluster sizes. They found that the ideal attributes and corresponding priority, for clustering, was first the number of nets localized, and then the reduction of terminal count. However, this method did not consider multi-level clustering, and the matching stage of the heuristic was very time-consuming.

There have been many methods used to form clusters in a circuit net list. Those methods

surveyed here each have advantages and disadvantages, but no one method is ideally suited for the standard-cell placement problem. Of all the methods examined, the MHEC method in hMetis displays most of the characteristics demanded in the standard-cell problem, apart from consideration of cluster sizes. However, there is nothing inherent in MHEC that would prevent such considerations from being implemented. In the next section, a new novel approach for circuit clustering is presented, based on weighted hyperedge clustering, which displays linear time-complexity, highly-connected clusters, high net localization, cluster size control, and good performance.

4.3 Clustering-based Standard-Cell Placement

4.3.1 Weighted Hyperedge Clustering

As we saw in the last section, little previous work has been done on the clustering-based placement problem, but many different ideas for clustering have been proposed that could have potential applications in placement. In particular, fast methods for clustering, based on linear-time measures of connectivity, proposed by Karypis et. al. [Kar97] showed high-quality results for the partitioning problem, while displaying many of the properties that are desirable in a clusterer for placement, namely speed, fairly even natural cluster sizes, and good circuit size reduction. In this section we describe a new clustering method based on this previous work but more suitable for the placement problem. As will be seen, the major addition to simple hyperedge clustering was the development of cluster size control.

The major problem with Karypis' MHEC method when applied to placement is that there is no explicit cluster width control. Sun and Sechen showed that for good quality placements, consistent cluster widths resulted in superior results [SunSec95].

In general, MHEC clusters consist of similar numbers of cells, typically between two and four [Kar97]. However, in the placement problem, a more important consideration is cluster *width*. A pair of clusters each consisting of two cells may have widths differing by an order of magnitude, and clusters with larger or smaller numbers of cells compound this problem. The result is that, while the sizes of the clusters tend to be consistent (and do not *have* to be consistent), the resulting

widths of clusters are almost certainly not.

In our approach, which we call a"Weighted Hyperedge Clustering", or WHEC, the approach is to explicitly limit cluster widths as they are formed. As seen in the pseudo-code in Figure 4.2, an upper and a lower width limit is determined based on the cell widths in the current hierarchal level. As a potential clustering of cells is examined, a new cluster is only created if the sum of the constituent cells' widths is between these width limits. This limitation on sizes prevents excessively large clusters from impeding improvers, yet still reduces the problem size.

Similar to the MHEC method, our method is divided into passes. In the first pass, cells hyperedges are greedily clustered together, but only if they are within width limits. In the second pass, remaining cells on hyperedges are also greedily clustered together. Finally, since it is possible that some cells are only connected to a cluster that is prohibited from clustering based on its size, a third pass is performed to assign any remaining cells to a new cluster.

We say that our method is "weighted" clustering, but in fact only the first pass is really size-limited. However, by observing the operation of the MHEC method, we know that this is where the largest clusters are formed. Since smaller nets are clustered first, the first pass creates mostly small clusters, resulting in clusters with generally small widths, but occasionally a large net will survive and result in a large cluster. By limiting the first pass sizes, these few large nets are avoided, and the second pass ensues. Cells on the smaller nets are clustered in the second pass, which have by now been diminished in total unclustered cell width, guaranteeing that by the time the larger nets are again visited, there are few unclustered cells remaining on them. The result is that huge clusters are eliminated, while still allowing a large amount of circuit size reduction.

Limiting sizes in the first pass only prevents huge clusters from forming. Therefore, the criteria for selecting size limits need not be complex. Observations have noted that any limit on the same magnitude of the original circuit, such as twice the average cell width, or twice the maximum cell width, give virtually identical results. In our investigation, twice the maximum cell width was used to limit sizes in the first pass. This encourages at most a doubling of the maximum cell size at each level of the hierarchy.

```
1.    Sort nets by increasing size
2.    For each sorted net
3.      If no cell on net is clustered
4.        If sum of cell widths on net is within limits
5.          Cluster all cells on net
6.        End If
7.      End If
8.    End For
9.    For each sorted net
10.     If sum of unclustered cell widths on net is within limits
11.       Cluster all unclustered cells on net
12.     End If
13.   End For
14.   For each cell in circuit
15.     If not clustered
16.       Create a new cluster from cell
17.     End If
18.   End For
```

Figure 4.2: Weighted Hyperedge Clustering

4.3.2 Implementation as a mapping function

The clustering task requires a good data structure for mapping cells-to-clusters and clusters-to-cells, the latter being useful for de-clustering. Assigning a cell to a cluster should be a constant-time operation, and traversing the entire structure, and destroying it, should be linear time in the size of the circuit.

The data structure used in this dissertation is shown in Figure 4.3. An index array is used for mapping cells to clusters, and an array of singly-linked lists is used for mapping clusters to cells.

This data structure allows a cell to be assigned to a cluster in constant-time, since adding a cell to the cell-to-cluster array, and adding a cell to the head of the cluster-to-cell list, are both constant-time operations. Determining if a cell is clustered and which cluster the cell is assigned to can also be done in constant-time. Also important is accessing the cluster-to-cell lists, since they must be traversed when de-clustering. Traversing a list is a linear-time operation, and examining the array of lists is also linear-time. However, since the union of all lists is equal to the original circuit list of cells, and no cell belongs to more than one cluster, examining all cluster-to-cell

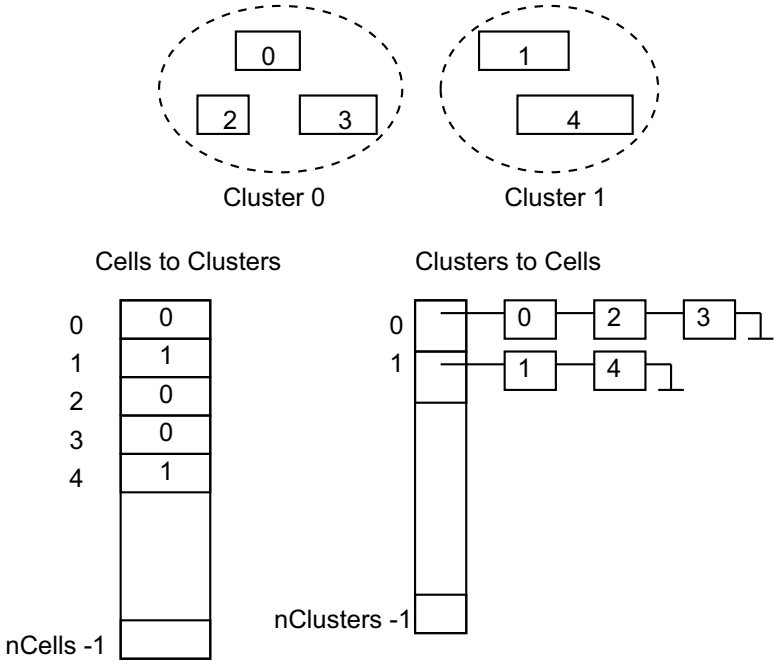

Figure 4.3: A Possible Mapping Implementation, showing organization of cells-to-clusters and clusters-to-cells structures.

lists during de-clustering can be done in linear-time, in the order of the number of cells in the de-clustered circuit!

Therefore, this data structure allows clustering and de-clustering, apart from sorting the net-list by size (which must be done only once in WHEC), to be done in linear-time.

4.3.3 Reduction of Nets

One of the primary goals of the clustering task is to reduce the number of nets in the clustered circuit. Regardless of the method used for clustering cells, reduction of the nets can proceed in a linear fashion once all cells have been assigned to clusters, and still produce both reduced-size nets and eliminate localized nets in the clustered circuit.

The method for doing this is illustrated in Figure 4.4. Originally Net 1 connects cells B and C, and Net 2 connects all three cells. Assuming that the clustering step clusters cells B and C together, Net 2 connects cell A with the new cluster. However, Net 1 connects the two cells that are now merged into a single cluster. At the new level of the hierarchy, Net 1 serves no purpose, since its wirelength cannot possibly be affected by any movement of clusters in the next level, and can be eliminated from the coarsened circuit's net list. It should be remembered at higher levels of the hierarchy that these nets still exist, and will be placed back into the net list during de-clustering. Removing these nets will remove their associated wirelength from the coarsened circuit. However, these nets will still connect the same modules, and will have a non-zero wirelength, so during de-clustering, these nets will add to the total circuit wirelength. For this reason, the wirelength of the top level circuit is not necessarily a realistic or attainable wire length for the original flat circuit.

Note also that Net 2 only requires a single connection to the new cluster to retain the connectivity information from the previous hierarchal level, so the two pins that used to connect cells B and C can be replaced by a single pin connected to the new cluster (Figure 4.4(C)).

The algorithm for reducing the number of nets is presented in Figure 4.5.

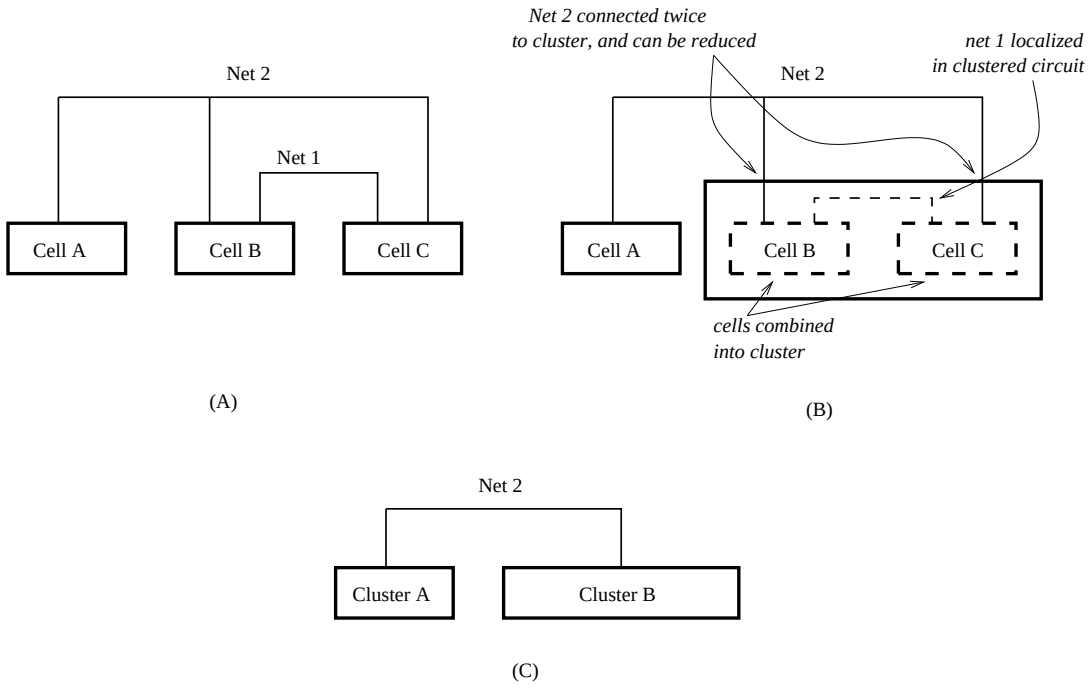

Figure 4.4: Nets are Reduced (Eliminated) when all pins are inside the cluster.

```
1.    For each net in hierarchy level n
2.      C = # of distinct clusters connected by net in level n + 1
3.      If C == 1
4.        Skip net
5.      Else
6.        Create new net in level n + 1
7.      End If
8.    End For
```

Figure 4.5: Net Reduction Algorithm

4.3.4 Pin Placement

In the partitioning problem, a net that is connected to more than one cell within a cluster in a flat circuit can be connected a single time to the cluster in the coarsened circuit without losing any information. This is a powerful feature of using clustering in partitioning since the time needed to evaluate a partitioning (by calculating a cost function) is proportional to the number of pins in the circuit.

In placement, the problem is much more complicated. The evaluation of a placement is based on 2-dimensional coordinates of pins in the circuit. It is important to account for all pin positions, since electrically-equivalent pins on a cell cannot be grouped together the way they can in partitioning. Ideally, when cells are clustered together, the pin positions on the cluster depend on the relative positions of the cells within the cluster. This presents a dilemma, since determining the relative placement of the cells presumes a knowledge of the ideal placement of the cells. However, we don't know this ideal placement, and we require a relative placement in order to determine a good placement for the clusters.

The approach taken in all past work in clustering-based placement (e.g. [MalGro89, SunSec95]) has been to assign all pin positions to the center of the cluster, as in Figure 4.6. The reason for this is to minimize the eventual increase in wirelength caused by flattening out the circuit. This increase is minimized when the error between the approximated pin positions and the eventual pin positions is small, which implies that cluster sizes (widths) should not be allowed to get too great, perhaps as a function of the row separation distance.

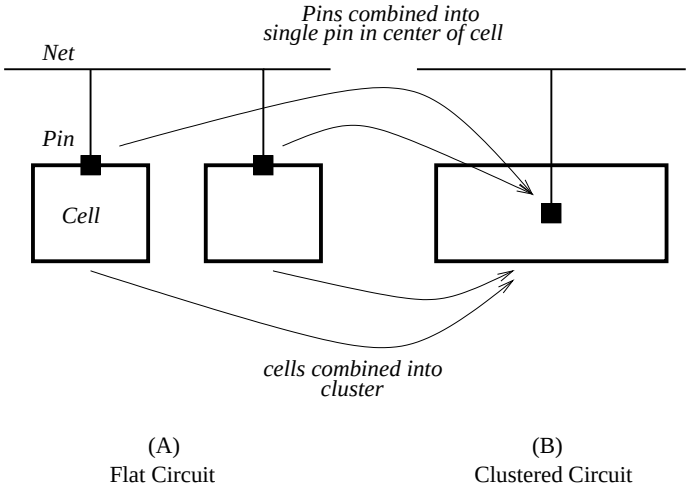

Figure 4.6: Pins are all placed at the center of a new cluster.

4.3.5 De-clustering

Legalization using FLATTEN

In this section, a very quick heuristic, FLATTEN, is presented which improves de-clustering quality slightly. Previous methods for clustering-based placement [SunSec95, MalGro89] flattened the circuit by placing the cells of a cluster randomly within the physical confines of the cluster at the previous hierarchal level. Since the cluster module is created with equivalent height and area as its cells, de-clustering a cluster does not change the size of the chip. Since relative positions between cells in a cluster were not considered at any clustered level of the hierarchy, they are not implied at the flattened level, and so some method must be used to determine legal relative positions for the flattened cells to occupy.

As the circuit is flattened some worsening of the solution is inevitable. This is because during the de-clustering phase, nets that were localized to a cluster must be added back to the flattened circuit as it is broken down into its constituent cells, increasing total wirelength.

Another source of wirelength increase during de-clustering is the approximation made for pin placement. The problem is that the pin placement in the clustered levels (i.e., overlapping cells with pins in the center of the cluster) is a poor approximation for the true pin positions in the

eventual relative placement of the cells. The top-level placement is made, assuming that the estimated pin locations are the same as the pin positions for corresponding cells in the original circuit. As the clusters are broken down, this difference is exposed, and the overall solution gets worse. The approximation error increases as the cluster size increases, since the difference between the possible real pin positions and the center of the cluster is greater. For this reason it is important to keep the maximum cluster size reasonably low.

To minimize the quality deterioration during circuit flattening, further improvement is performed on the circuit at each flattening stage, using a very localized search heuristic. A greedy method for reducing the legalization error, which we call FLATTEN, is shown in Figure 4.7.

In the FLATTEN heuristic, the average position of all connected pins is calculated for each cell in a cluster (indicated by the light gray circle and square in Figure 4.7(A)). The cells within the cluster are then sorted by their average pin x-coordinate, and given a relative order as they are flattened (in Figure 4.7(B), cells A and B are swapped in accordance with their average pin positions). This method does not search for a reduced cost of the overall placement, and no HPWL is computed, but instead places the cells in a location more tenable for local improvement. A local improvement step is still required after this quick legalization step.

Local Improvement

As previously mentioned, the majority of placement improvement work is targeted for the highest level of the hierarchy, where it can take advantage of the reduction in complexity. If the clustering method and improver perform well, the placement achieved at the highest level should be of very high quality. However, as the circuit is clustered and improvements made at the top level of the hierarchy, many approximations must be made. Many decisions must be made to reduce the complexity of the problem while still representing the flat circuit accurately, such as estimating pin positions and removing large nets.

These approximations made while clustering lead to significant quality deterioration during de-clustering, as approximations made at the top hierarchal level "trickle-down" through each de-clustering step. Localized improvers must be used to minimize this deterioration as it is introduced

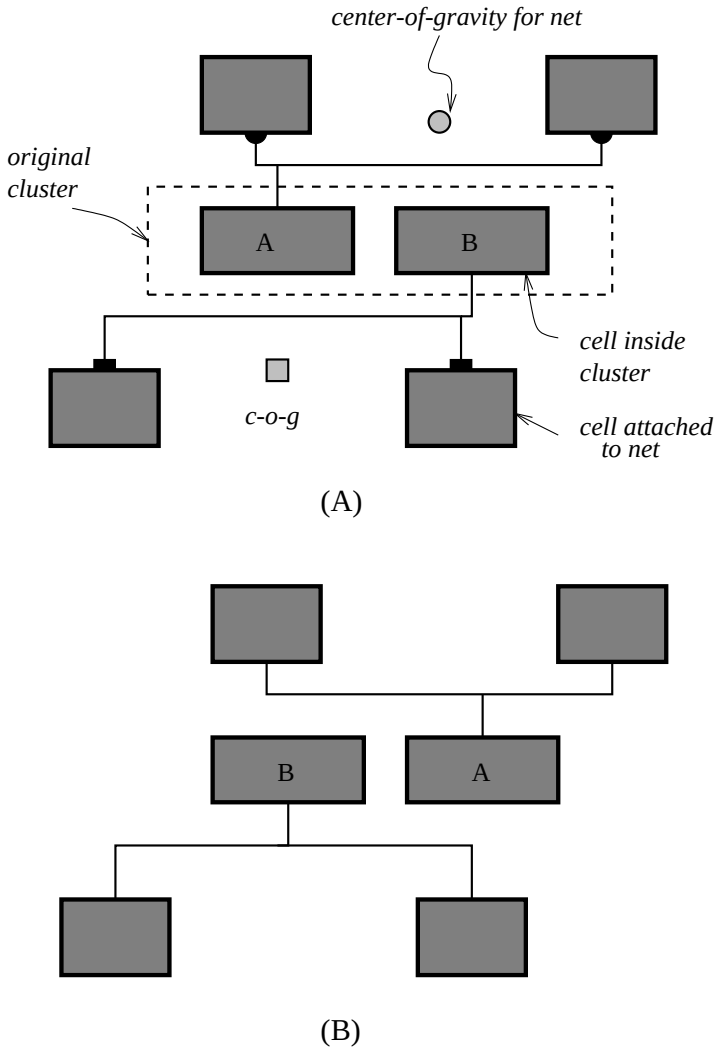

Figure 4.7: Cells within a cluster are given relative placements according to their sorted average pin positions

at each de-clustering phase, but because the top-level solution is (presumably) of high quality, the improvers used during de-clustering should be able to find a local minimum with little effort. Recall that as the circuit is de-clustered, the problem size increases; work done at lower levels of the hierarchy take longer than that at upper levels, so it is prefereable to distribute as much work as possible to the upper levels.

What is needed is an improver that excels at greedily improving a solution with very localized moves, in as short a time as possible. An excellent candidate for this is the utility-based improver, UTILITY, proposed in chapter 3. This is the method that was used in this investigation.

4.3.6 Top-Level Improvement

The majority of the improvement work should be done at the highest level of the hierarchy, to take advantage of the reduced search space provided by clustering the circuit. Quality of the solution at this point is, obviously, mostly a factor of the improvement heuristic quality. However, the *potential* for improvement is limited by the clustering method. A good clustering method will reduce the original circuit search space to a similar, but much smaller, search space, with many cells in the placement already moved in proximity to one another through the clusters they form. In a clustering such as this, a good improvement heuristic will attain a high-quality solution.

By contrast, a poor clustering method may reduce the size of the search space, but the search space will not be very similar to the original space. This means that what may be a very good solution in the reduced search space, will not translate to a good solution in the flat search space. Therefore, two clustering methods that reduce the search space by the same degree are not necessarily equivalent. This premise is what we base our observations on; the quality of a clustering method can only be determined from the final solution quality.

The ARP global placer is the initial placement heuristic used at the top level of the hierarchy, which is known to provide good global solutions in reasonable amounts of time [Eta99]. After the cells in the global solution are assigned legal positions, a tile-based iterative improvement method [Ken97] is used to correct for errors induced by the legalization process. Together, these placement heuristics adequately explore the search space of the top-level circuit placement.

4.4 Results

The quality of a clustering method can only be measured by comparing final placement quality to that of flat placement heuristics. Even in a relatively simple clustering method such as WHEC, however, there are numerous factors that can affect performance. We can't control factors such as circuit connectivity, but we can characterize a clustering method's behavior.

Section 4.4.1 discusses some important properties of the benchmark circuits used for testing the performance of our clustering technique.

Sections 4.4.2 and 4.4.3 show the performance of the WHEC clustering heuristic. Section 4.4.2 summarizes the performance of our hierarchal placement method to a similar flat placement method, while section 4.4.3 compares our clustering technique to several other similar clustering methods.

Sections 4.4.4 and 4.4.5 show the affect of varying factors on clustering heuristic performance. Section 4.4.4 illustrates the changes in circuit connectivity and cell width distribution after clustering, linking clustering performance to desirable and avoidable circuit properties. Section 4.4.5 shows the affect of varying clustering depth on solution quality, showing how cluster depth also affects circuit statistics. Finally, section 4.4.6 summarizes the performance of the FLATTEN de-clustering heuristic.

4.4.1 Test Circuits

The same benchmark circuits used to test the utility-based improver in chapter 3 were used to test the clustering heuristics, and are presented in section 2.5.

There are several observations that can be made about the benchmark circuits, that play a role during clustering and placement. The benchmarks circuits in Table 2.5 have widely varying pad distributions. This has little impact on iterative improvement techniques, but it can have a profound influence on global placers that use the placement of pads to "spread" the placement of cells in the chip core [EtaAre99]. When a global placer is used, circuits with highly symmetrical pad positions lead to better quality solutions, since legalizing the global solution is easier. Circuits with highly assymetrical pad positions have lower-quality solutions after the global placement step,

and are therefore more difficult to improve by local search techniques due to increased legalization error. The Struct and Biomed benchmark circuits are particularly susceptible to this problem, while circuits such as avq.small and avq.large are fairly symmetrical and are less susceptible.

Cell widths also vary greatly among the various benchmark circuits, but of particular interest is the standard deviation of cell widths. The circuits can be roughly grouped by cell width deviation into those with a low deviation, below 20 or so, and those above it. A low deviation value means that most cells are close to the mean width, and so the cell sizes are fairly evenly sized. Notice that, of the large circuits, both Industry2 and Industry3 have large standard deviation values.

Examining the module degree and net size statistics points out the interesting property that all circuits, no matter how complex, have very similar connectivity. Notice that despite the very large range of maximum net sizes in Table 2.2, the mean and standard deviation vary over a very small range. This indicates that the average large digital circuit (which we are placing in standard-cells) consists largely of three-port devices (such as a 2-1 NAND gate) with an average fan-out of 2, as might be expected.

4.4.2 WHEC Clustering Performance

Table 4.1 summarizes the performance of our hierarchal placement method using WHEC clustering, compared with a comparable flat placement heuristic. The results for clustering-based placement were obtained by running the improvement heuristic with three levels of clustering, using the ARP global placer [EtaVan98] and the TILE iterative improver [Ken97] at the top hierarchal level, and the UTILITY iterative improver (see Chapter 3) during de-clustering. In the flat placement heuristic (columns labeled "FLAT" in the results), the same placement method was used as was used in the clustering heuristic, but no clustering (or de-clustering) was performed. Since UTILITY was used in the hierarchal heuristic, it was also used as a final post-processing step in the flat heuristic.

Both the WHEC-clustered and flat improvement results were attained using 40 passes of TILE (examining each cell in the circuit 20 times per pass), and 100 passes of UTILITY examining 20% of the utility list each pass. For this and all other results, wirelength is measured in meters and

Circuit	Wire Length *Flat*	Wire Length *WHEC*	Wire Length Reduction	Time *Flat*	Time *WHEC*	Speedup
fract	0.035	0.035	0%	10	4.4	2.3
prim1	0.85	0.89	-4.7%	141	50	2.8
struct	0.44	0.42	+4.5%	205	52	3.9
Sm. average	-	-	-0.1%	-	-	3.0
ind1	1.68	1.61	+4.2%	618	270	2.3
prim2	4.24	4.27	-0.7%	624	224	2.8
bio	2.21	2.25	-1.8%	1381	391	3.5
Med. average	-	-	+0.6%	-	-	2.9
ind2	20.0	21.0	-5.0%	3411	2061	1.7
ind3	54.0	56.4	-4.4%	4453	1624	2.7
avq.small	9.81	7.14	+27.0%	6812	2385	2.9
avq.large	11.41	8.36	+26.7%	7917	2857	2.8
Lg. average	-	-	+11.0%	-	-	2.5
average	-	-	+4.6%	-	-	2.8

Table 4.1: Wirelength and Run-Time Comparison, FLAT Versus WHEC

time is measured in user time on a Sun Ultra10 workstation with 512MB of RAM.

The results in Table 4.1 show that the WHEC-based heuristic achieved similar-quality results to, but an average of three times faster than, those of the flat placement heuristic. The results also show that larger circuits are improved to a greater degree than smaller circuits, while run-time speedup is fairly consistent regardless of circuit size. For the largest circuits which were very responsive to clustering-based placement, avq.small and avq.large, the WHEC heuristic achieved a 27% improvement in wire length, while taking only 36% of the flat heuristic run-time, showing that, particularly for large circuits, WHEC is very effective for standard-cell placement.

4.4.3 Clustering Method Comparison

Tables 4.2, 4.3, and 4.4 summarize the results of our new clustering heuristic, compared with EC and MHEC clustering heuristics proposed by Karypis et. al. [Kar97], and a non-clustered (flat) improvement heuristic. MHEC was used for comparison since it is the basis for our WHEC method, and produced excellent results when applied to the partitioning problem. The EC method was used as another basis for comparison, since it represents a highly random and simplistic clustering method. The flat heuristic results are repeated for completeness. The results were obtained by

Circuit	Flat	EC	MHEC	WHEC	WHEC(2)
Fract	0.035	0.044	0.036	0.036	0.035
Prim1	0.85	1.23	0.91	0.89	0.87
Struct	0.44	0.73	0.48	0.42	0.41
Sm. Agg.	1.33	2.36	1.43	1.35	1.32
Ind1	1.68	1.86	1.65	1.62	1.63
Prim2	4.24	5.43	4.54	4.28	4.13
Bio	2.21	2.62	2.93	2.26	2.23
Me. Agg.	8.13	9.91	9.12	8.16	7.99
Ind2	20.0	22.8	20.4	21.0	20.6
Ind3	54.0	72.8	60.5	56.4	53.8
Avq.small	9.81	8.43	7.34	7.14	8.85
Avq.large	11.41	8.58	8.71	8.36	9.66
Lg. Agg.	95.22	112.61	96.95	92.90	92.91
Agg. WL	104.62	124.88	107.50	102.41	102.22

Table 4.2: Wirelength Comparison, WHEC Versus EC and MHEC

running the improvement heuristic with three levels of clustering, using ARP and TILE at the top hierarchal level and UTILITY during de-clustering. The column marked WHEC(2) shows the results when cluster sizes are limited during the second pass of the WHEC heuristic, in addition to the first pass as in "standard" WHEC. Clustered reductions in net and cell counts in Table 4.4 are in relation to the original unclustered benchmark circuits in section 2.5.

Compared to EC and MHEC, WHEC achieved the best solution quality of all methods, and usually improved on the flat solutions, but took longer than the other clustering methods. The EC method is characterized as the fastest method, due to the fact that it reduced the number of cells in the circuit the most. However, the results attained were not as good as the hyperedge based methods, suggesting that quality is not necessarily dependent on cell-count reduction alone. The MHEC method results fell between EC and WHEC in quality and speed. It reduced the number of nets the most, which is logical since clustering was performed by collapsing hyperedges (nets). Since WHEC was a constrained version of MHEC, it clustered fewer cells and and collapsed fewer nets that MHEC and EC.

The WHEC(2) method warrants special consideration. This method shows the importance of balanced cluster sizes during placement. For the circuits with uneven cluster sizes (see section 4.4.4

Circuit	Flat	EC	MHEC	WHEC	WHEC(2)
fract	10	2.3	3	4.4	4.6
prim1	141	22	36	50	55
struct	205	44	44	52	145
Sm. Agg.	356	68	83	106	205
ind1	618	131	246	270	300
prim2	624	135	193	224	270
bio	1381	318	543	391	578
Me. Agg.	2623	584	982	885	1148
ind2	3205	829	2056	2061	1805
ind3	4453	943	1892	1624	2185
avq.small	6812	1289	2133	2385	3069
avq.large	7917	1512	2510	2857	3551
Lg. Agg.	22387	4573	8591	8927	10610
Agg. WL	25366	5225	9656	9918	11963

Table 4.3: Run Time Comparison, WHEC Versus EC and MHEC

Circuit	EC				MHEC				WHEC			
	Mods	Red.	Nets	Red.	Mods	Red.	Nets	Red.	Mods	Red.	Nets	Red.
Fract	50	66 %	107	27 %	80	46 %	99	33 %	88	41 %	107	27 %
Primary1	215	74 %	608	33 %	324	61 %	457	49 %	389	53 %	509	44 %
Struct	382	80 %	1245	35 %	346	82 %	612	68 %	511	74 %	780	59 %
Sm. Avg.	-	73 %	-	32 %	-	63 %	-	50 %	-	56 %	-	43 %
Industry1	1228	60 %	1677	32 %	1740	44 %	1332	46 %	1892	39 %	1424	43 %
Primary2	602	80 %	2162	29 %	1068	65 %	1458	52 %	1311	57 %	1591	47 %
Biomed	1654	75 %	2668	54 %	1657	75 %	1712	70 %	2047	69 %	1733	70 %
Me. Avg.	-	72 %	-	38 %	-	61 %	-	56 %	-	55 %	-	53 %
Industry2	3073	76 %	8657	35 %	4836	62 %	6745	50 %	5917	53 %	7516	44 %
Industry3	2800	82 %	15839	28 %	4399	71 %	12314	44 %	6427	58 %	13806	37 %
Avq.small	3631	83 %	10836	51 %	5969	73 %	7987	64 %	7258	67 %	9051	59 %
Avq.large	4363	83 %	11816	53 %	6733	73 %	884	97 %	8434	67 %	10204	60 %
Lg. Avg.	-	81 %	-	42 %	-	70 %	-	64 %	-	61 %	-	50 %
Average	-	76 %	-	45 %	-	65 %	-	57 %	-	58 %	-	49 %

Table 4.4: Cell and net reduction after three levels of clustering, using EC, MHEC, and clustering heuristics

Circuit	Cell Width				Module Degree			Net Size		
	Min	Max	$\bar{x}$	σ	Max	$\bar{x}$	σ	Max	$\bar{x}$	σ
Fract	16	312	149.5	100.4	18	9.5	5.0	9	2.55	1.26
Primary1	30	1340	598.3	386.7	26	12.2	6.6	17	2.82	2.23
Struct	16	240	154.5	71.3	15	9.2	4.2	17	2.42	2.13
Industry1	12	396	173.6	93.5	25	11.0	5.8	219	3.06	7.09
Primary2	30	1650	525.4	326.6	39	13.9	7.0	32	3.23	3.28
Biomed	16	408	144.6	97.5	22	7.7	4.2	42	3.35	4.34
Industry2	32	1536	388.0	272.9	42	11.1	7.4	424	3.37	9.03
Industry3	56	1224	581.2	259.5	53	17.4	8.8	214	2.68	2.72
Avq.small	16	488	202.8	103.0	23	10.0	4.3	343	2.42	5.51
Avq.large	16	488	186.3	103.3	21	8.7	4.0	336	2.38	5.28

Table 4.5: Size and Connectivity Statistics, EC Clustering

below), this method performed better than the other methods, and is the only method which beat the flat results for the industry3 benchmark. WHEC(2) is instructional to examine because it shows that rigorously restricting cluster sizes does have a beneficial effect on circuits where the other methods have difficulty. However, WHEC(2) performed very poorly for most circuits - it will not be included in further comparisons.

All clustering methods had difficulty improving some circuits, particularly industry2, industry3, prim1, and prim2. By contrast, the avq circuits tended to be highly responsive to improvement by clustering, and results obtained by all methods were much better than those achieved using the flat heuristic. This observation suggests that circuit topology plays an important role in clustering-based improvement, with some topologies being more or less responsive to clustering-based placement. These issues are discussed more fully in the next subsection.

4.4.4 Effect of Cell Statistics

Tables 4.5, 4.6, and 4.7 show cell width and connectivity distributions after circuit clustering with EC, MHEC, and WHEC clustering heuristics, respectively. The results were obtained by running the improvement heuristic with three levels of clustering, using ARP and TILE at the top hierarchal level and UTILITY during de-clustering.

Comparing Tables 4.5, 4.6, and 4.7 with the improvement results in Table 4.2 shows that there

Circuit	Cell Width				Module Degree			Net Size		
	Min	Max	$\bar{x}$	σ	Max	$\bar{x}$	σ	Max	$\bar{x}$	σ
Fract	16	336	97.2	97.2	15	6.0	4.0	13	3.0	2.0
Primary1	30	2110	387.3	353.2	21	7.3	4.5	18	3.8	3.0
Struct	16	336	174.2	112.9	11	6.7	2.8	17	3.2	2.8
Industry1	12	374	83.8	70.3	25	5.7	3.7	284	4.3	10.5
Primary2	30	1660	290.5	244.9	45	7.3	6.1	33	4.7	4.4
Biomed	16	3912	144.2	212.5	89	6.8	5.5	390	4.7	10.7
Industry2	32	2912	258.7	236.0	56	7.4	5.6	487	4.5	12.5
Industry3	56	3192	375.7	292.8	89	10.8	7.9	285	3.3	3.7
Avq.small	16	1064	145.1	104.0	59	6.9	3.8	399	3.1	7.0
Avq.large	16	1104	140.4	99.1	50	6.4	3.5	376	3.1	6.4

Table 4.6: Size and Connectivity Statistics, MHEC Clustering

Circuit	Cell Width				Module Degree			Net Size		
	Min	Max	$\bar{x}$	σ	Max	$\bar{x}$	σ	Max	$\bar{x}$	σ
Fract	16	336	99.7	96.4	16	6.2	4.1	13	3	2.02
Primary1	30	1640	369.5	348.6	25	7.1	4.7	18	3.80	3.06
Struct	16	336	164.9	98.1	11	6.7	2.6	17	3.11	2.86
Industry1	12	410	83.4	67.3	22	5.7	3.5	284	4.33	10.5
Primary2	30	1560	285.2	234.1	45	7.2	5.9	33	4.78	4.47
Biomed	16	720	127.6	73.2	25	6.1	2.8	29	4.75	5.55
Industry2	32	2272	246.8	210.2	40	7.1	5.1	491	4.63	12.7
Industry3	56	3192	370.4	278.4	82	10.5	7.5	293	3.37	3.82
Avq.small	16	472	134.9	94.1	20	6.3	3.1	393	3.12	7.03
Avq.large	16	504	131.0	90.2	21	5.9	2.8	376	3.08	6.57

Table 4.7: Size and Connectivity Statistics, WHEC Clustering

is a high correlation between clustering-based improvement and cell width distribution. Circuits with high cell width deviations, such as industry2 and industry3, were more difficult to improve by clustering methods than those circuits with lower deviations, such as avq.small and avq.large.

All three clustering methods had a difficult time maintaining even cluster sizes for most benchmarks, compared to the flat circuit cell width deviations in Table 2.2 (in section 2.5). However, the cell width deviation is much lower when using WHEC than it is when using EC or MHEC. Since the results using WHEC were substantially better, we can conclude that evenly distributed cluster sizes are highly desirable, since they have a significant impact on a circuit's capacity to be placed well.

4.4.5 Clustering Depth

Figures 4.8, 4.9 and 4.10 show the effects of different clustering depths on solution quality using different clustering methods, as measured by total wirelength. Results were obtained by running the benchmarks with between one and six levels of clustering, using ARP and TILE at the top hierarchal level, and UTILITY during de-clustering. The three circuits shown represent the small, medium, and large circuits, respectively.

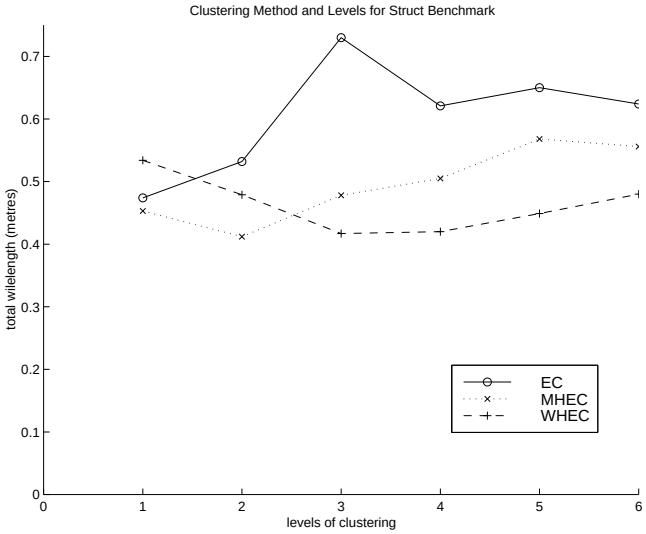

Figure 4.8: Wirelength Versus Clustering Depth, struct Benchmark

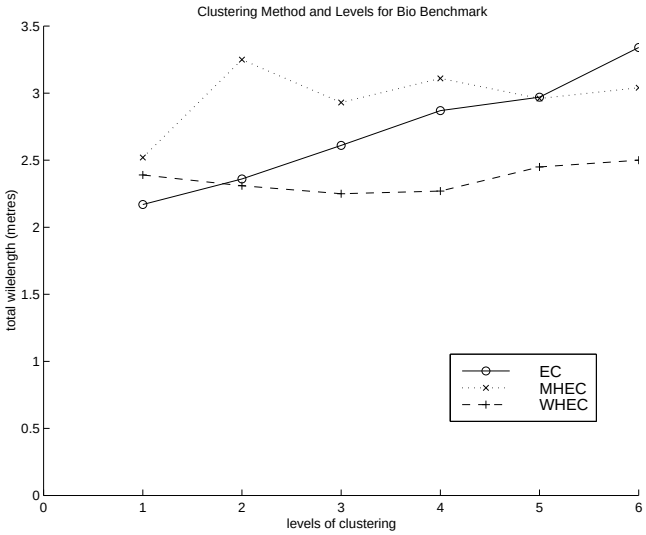

Figure 4.9: Wirelength Versus Clustering Depth, bio Benchmark

The results clearly show that, for WHEC, three levels of clustering gives good results for all sizes of circuits. For all clustering depths greater than two, WHEC out-performed both EC and MHEC for all circuit sizes. For small circuits, EC and MHEC generally give better results for lower levels of clustering than WHEC, but the solutions obtained were still inferior to the flat solutions. For medium and large circuits, WHEC is clearly preferable.

Figures 4.8, 4.9 and 4.10 also show that the WHEC method is fairly insensitive to increases in clustering depth. This is in stark contrast to the other clustering methods, EC and MHEC, which are much more sensitive to changes in clustering depth. For the avq.large benchmark (a large circuit) the ideal clustering depth was three levels, but increasing the clustering depth to four levels only increased wirelength by an average of 3%, versus a 10% increase when EC was used.

4.4.6 FLATTEN De-clustering Heuristic

Table 4.8 shows the effect of using the FLATTEN de-clustering heuristic, described in section 4.3.5, used during the de-clustering phase for different clustering depths. Results were obtained by using

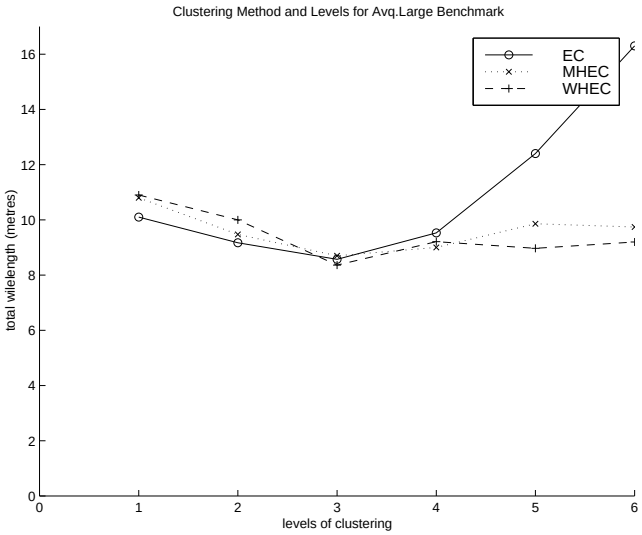

Figure 4.10: Wirelength Versus Clustering Depth, avq.large Benchmark

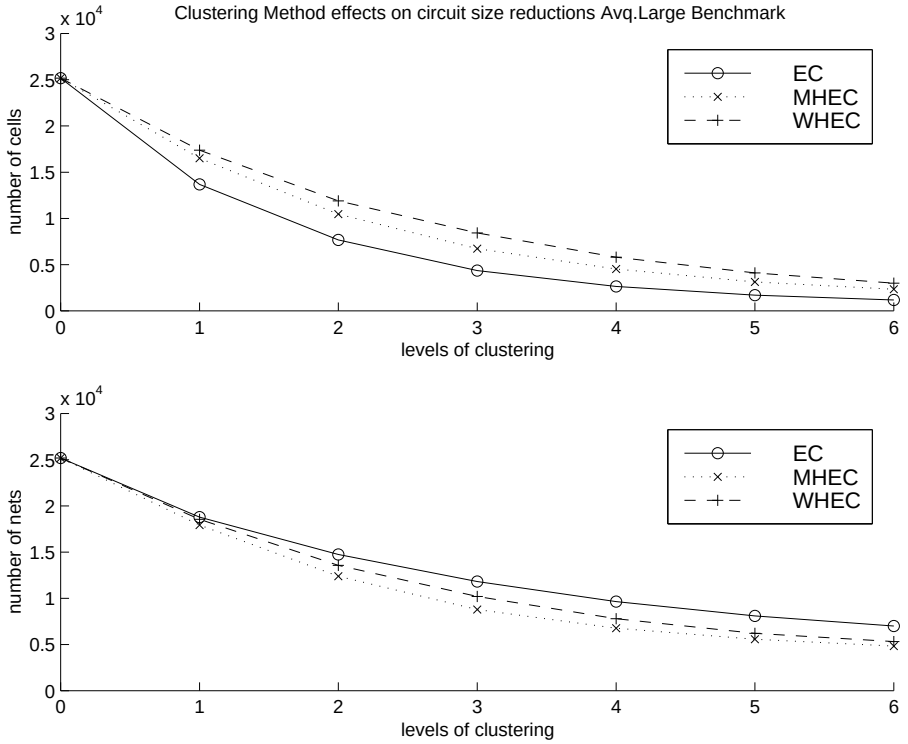

Figure 4.11: Clustering method effects on circuit size reduction, avq.large Benchmark

Circuit	One	Imp	Two	Imp	Three	Imp
fract	42654	0.97%	50586	1.71%	49870	2.50%
prim1	1131245	-0.78%	1143974	-0.23%	1209388	0.09%
struct	776255	0.79%	520059	3.08%	486149	7.09%
Sm. avg	-	0.32%	-	1.52%	-	3.23%
ind1	2064284	0.09%	2068205	0.25%	2085678	0.43%
prim2	5843413	0.08%	5589677	0.51%	5924559	0.76%
bio	3008984	0.01%	2845382	0.95%	2751993	1.29%
Med. avg	-	0.06%	-	0.57%	-	0.83%
ind2	28666972	0.09%	28035935	0.02%	29005368	0.14%
ind3	72989736	0.28%	82342961	0.34%	90299540	0.40%
avq.small	13739141	-3.82%	11714924	0.54%	9920093	1.56%
avq.large	15000070	-0.44%	12096544	7.67%	11209297	0.97%
Lg. avg	-	-0.97%	-	2.14%	-	0.77%
average	-	-0.59%	-	1.41%	-	1.60%

Table 4.8: Performance of FLATTEN de-clustering placement heuristic, after one, two, and three levels of clustering

the ARP global placer at the top hierarchal level, without any iterative improvement, and then using only FLATTEN during de-clustering, again without any iterative improvement. The reason for not using iterative improvement is that the search space for the FLATTEN heuristic is highly local, since cells within a cluster are only placed within the physical confines of the cluster. This means that cell positions can only change by a maximum of half the original cluster width (see section 4.3.5), so little improvement is expected during the fast FLATTEN heuristic, when compared to other improvement techniques. In order to emphasize this fact, no iterative improvers were used. Results using the FLATTEN heuristic were compared against clustered results using only ARP at the top hierarchal level and then de-clustering, placing cells randomly within the physical confines of their clusters.

Average improvement using the FLATTEN heuristic under these circumstances was between 1-2% for most benchmarks, and had the most effect on large circuits at higher clustering depths. As expected, little improvement was actually obtained, but the heuristic has a linear time complexity, so no noticeable difference in execution time was used to gain the improvement, showing that the FLATTEN heuristic is useful.

Circuit	Wire Length *No Imp.*	Wire Length *UTILITY*	Wire Length Reduction	Time *No Imp.*	Time *UTILITY*	Time Increase
fract	0.037	0.035	5.4 %	3.7	4.4	18.9 %
prim1	0.91	0.89	2.2 %	44	50	13.6 %
struct	0.44	0.42	4.5 %	40	52	30.0 %
Sm. average	-	-	4.0 %	-	-	20.8 %
ind1	1.65	1.61	2.4 %	237	270	13.9 %
prim2	4.37	4.27	2.3 %	196	224	14.2 %
bio	2.34	2.25	3.8 %	330	391	18.5 %
Med. average	-	-	2.8 %	-	-	15.5 %
ind2	22.2	21.0	5.4 %	1908	2061	8.0 %
ind3	58.0	56.4	2.8 %	1473	1624	10.3 %
avq.small	7.32	7.14	2.5 %	2135	2385	11.7 %
avq.large	8.49	8.36	1.5 %	2453	2857	16.5 %
Lg. average	-	-	3.0 %	-	-	11.6 %
average	-	-	3.2 %	-	-	15.5 %

Table 4.9: Effectiveness of using the UTILITY improver during circuit de-clustering

4.4.7 UTILITY De-clustering Improvement

Table 4.9 shows the effectiveness of the UTILITY heuristic used during de-clustering. Both sets of results were attained by applying three levels of clustering using WHEC, then improving the circuit with ARP and TILE at the top hierarchal level. The columns labeled "UTILITY" applied 100 iterations of the UTILITY heuristic after *emph* de-clustering step, while the columns labeled "No Imp." were not improved at all during de-clustering.

As the results show, there was an average 3.2% difference in total wirelength when not using improvement during de-clustering, but using the UTILITY improver increased overall run-time by an average 15.5%, showing that using de-clustering improvement can be very important for minimizing solution quality deterioration during de-clustering.

4.5 Summary

The WHEC method of clustering was shown to be very effective when applied to the standard-cell placement problem. Two other methods of clustering, edge clustering and modified hyperedge

clustering proposed by Karypis et. al. [Kar97] where shown to perform well for the placement problem, but the size-limiting feature of WHEC provided results superior to the other methods.

Results obtained using the WHEC clustering method were very close in quality to those of a flat heuristic. For large circuits, using clustering actually provided a 11% *improvement* over the flat solutions. However, the real advantage of using circuit clustering is the reduction in execution time allowed by reducing the problem size. Compared to the flat heuristic, the clustered solutions were attained an average of three times faster. In addition, the clustering and de-clustering of the circuit took only a few seconds, even on the largest benchmarks, proving that simple and fast clustering techniques can achieve large gains versus a flat technique.

A high correlation was shown to exist between evenly-distributed cluster sizes and final solution quality. When comparing different clustering methods, the circuits with the more evenly-distributed cluster sizes had better final results than those with more uneven sizes. When the cluster sizes formed were very uneven, final solution quality was usually inferior to those achieved using the flat placement heuristic.

It was shown that, for WHEC, three levels of clustering gave good results for all sizes of circuits. For all clustering depths greater than two, WHEC out-performed both EC and MHEC for all circuit sizes. For small circuits, EC and MHEC generally give better results for lower levels of clustering than WHEC, but the solutions obtained were still inferior to the flat solutions. For medium and large circuits, WHEC is clearly preferable. The WHEC method was shown to be less sensitive to increases in clustering depth than other clustering methods examined. For the avq.large benchmark (a large circuit) the ideal clustering depth was three levels, but increasing the clustering depth to four levels only increased wirelength by an average of 3%, versus a 10% increase when EC was used.

Chapter 5

Conclusions

Two new approaches to dealing with the high complexity of ASIC design have been presented. A novel utility-based iterative improvement heuristic and a new hierarchal clustering heuristic were shown to provide excellent characteristics for reducing the execution time of standard-cell placement while improving results, when compared to good existing approaches.

When performing a greedy search close to a local minimum in the search space, the majority of moves within a solution's neighborhood do not improve the solution quality. A novel new approach to overcome the problem of random move selection in a deterministic (greedy) search heuristic is to use *utility-theory* to find a local minimum quickly by ranking moves on an estimate of their proximity to an optimal location, and then choose moves that are statistically more likely to improve than if the moves were chosen randomly.

When applied to the standard-cell placement problem, a cell's *utility* is a measure of how close to optimality the cell's placement is. By concentrating a search's computational effort on cells with worse-than-average placements, greater gain can be achieved by the utility-based search with less effort than that which a randomized move-selection method could achieve in the same time. In this way, it is possible to reduce the computational impact of examining moves that will be rejected.

When applied to non-hierarchal placement, it was shown that using utility as a measure of

placement quality was effective. Improving the cell utility of poorly-placed cells improved the total wire length of the solution, and solution convergence time was dramatically increased with smaller utility list search depths, so that "good" solutions were found quickly. However, when compared to a good deterministic search heuristic as a primary improvement heuristic, the results using utility were not competitive. The convergence time for the utility-based heuristic was too short to adequately examine the search space, and so the solutions were of very low-quality.

Utility-based improvement was shown to be best suited for use as a fast post-processing improvement step, after significant improvement has already been made by another improvement method. It was most effective when used on a placement after other improvement techniques had converged, and the utility-based search was able to rapidly seek a local minimum in the search space. Because utility-based improvement is highly deterministic, quickly finding a local minimum, it is well suited to achieve any *post-processing* improvement.

Window size was shown to have a significant impact on the performance of our heuristic. However, a relation between circuit statistics and window size was not found. As well, compromises were made to select a good utility list search depth, and number of improvement passes that would produce good results for all circuits. Future directions for research into utility-based improvement could try to develop an adaptive heuristic that finds ideal utility list depth, number of passes, and window size parameters for different circuits. As well, incorporating a mix of gain estimation in earlier search stages, and exact gain calculation during later search stages, might produce good results.

Overall, utility-based searching is very effective, when applied to the right kind of problem. Due to its highly-localized search space and rapid convergence properties, it should only be applied when a rapid solution is needed regardless of high-quality, or when the search is expected to be in the vicinity of a good local minimum in the search-space.

The size of ASIC design problems is increasing at a substantial rate, and methods for dealing with highly complex problems are needed. One method for dealing with complex systems is to narrow a search to localized regions of the search space. Utility-based placement is one excellent example of this. Another method is to reduce the complexity of the problem itself, by restructuring

it into a smaller form through *circuit clustering.*

Our new hierarchal clustering method, weighted hyper-edge clustering, was shown to be very effective when applied to the standard-cell placement problem. Although performance was fairly good for small- and medium-sized problems, performance was highest on the largest benchmark circuits.

Two other methods of clustering: edge clustering, and modified hyper-edge clustering, both proposed by Karypis et. al. ([Kar97]), where shown to perform well for the placement problem, but the size-limiting feature of our clustering method provided results superior to the other methods.

Results obtained using our clustering method were very close in quality to those of a flat heuristic, actually improving the solution for the largest benchmarks. Compared to the flat heuristic, the clustered solutions were attained an average of three times faster and the clustering and de-clustering of the circuit took only a few seconds, even on the largest benchmarks, proving that simple and fast clustering techniques can achieve large gains versus a flat technique.

A high correlation was shown to exist between evenly-distributed cluster sizes and final solution quality. When comparing different clustering methods, the circuits with the more evenly-distributed cluster sizes had better final results than those with more uneven sizes. When the cluster sizes formed were very uneven, final solution quality was usually inferior to those achieved using the flat placement heuristic. Therefore, maintaining even cluster sizes is very important to a clustering heuristic.

It was shown that, for our clustering method, three levels of clustering gave good results for all sizes of circuits, and gave consistently superior results at greater clustering depths to those of other clustering methods examined. While our method did not perform as well as edge-clustering or modified hyper-edge clustering methods on small, our method is clearly preferable for medium and large circuits. Weighted hyper-edge clustering was shown to be much less sensitive to increases in clustering depth than other clustering methods examined, making it more robust for practical implementation, when experimenting with different cluster depths is not feasible. For the avq.large benchmark (a large circuit) the ideal clustering depth was three levels, but increasing the clustering depth to four levels only increased wire length by an average of 3%, versus a 10% increase when

edge-clustering was used.

A fast method of reducing the impact of de-clustering was shown to be effective. The method functions by finding a relative placement of the cells in a cluster based on the sorting the center-of-gravity of each cell's connected pins. The method operates in linear-time complexity, while typically achieved 1-2% improvement.

Future work in the clustering problem should focus on reducing the unevenness of cluster sizes. However, some circuits were insensitive to cluster size deviation, while others were very unforgiving. A modification to weighted hyper-edge clustering would perform very well if some adaptive technique for modifying weighting factors to accommodate these circuits could be found. The optimum clustering depth was found to be slightly different for different benchmarks. Attempts to find a correlation between clustering depth and circuit statistics was unsuccessful. Developing an adaptive technique to determine the ideal depth, it could save significant time and effort on the part of the designer using this heuristic.

Bibliography

[SaiYou95] S. M. Sait and H. Youssef, "VLSI Physical Design Automation," IEEE Press, New Jersey, 1995.

[Ger99] S. H. Gerez, "Algorithms for VLSI Design Automation," John Wiley & Sons, New York, 1999.

[BelElm95] A. Bellaouar and M. I. Elmasry, "Low-Power Digital VLSI Design," Kluwer Academic Publishers, Boston, 1995.

[GarJoh79] M. R. Garey and D. S. Johnson, "Computers and Intractability," Freeman, San Francisco, 1979.

[FidMat82] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. Design Automation Conference*, 1982, pp. 175-181.

[AlpKah95] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey," *Integration: The VLSI Journal*, 1995, pp. 1-81.

[AlpHua98] C. J. Alpert, J. H. Huang and A. B. Kahng, "Multilevel Circuit Partitioning," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* 1998, pp. 655-667.

[CohPar87] J. P. Cohoon and W. D. Paris, "Genetic Placement," *IEEE Trans. Computer-Aided Design*, Vol. CAD-6, no. 6, November 1987, pp. 956-964.

[Beh99] L. Behjat, "A Concentric Placement Approach for Standard Cell Layout," M.A.Sc. Thesis, University of Waterloo, Waterloo, Ontario, 1999.

[SunSec95] W.-J. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol14, no3, march 1995, pp. 349-359.

[KirGel83] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, no. 4598, 13 May 1983, pp. 671-680.

[Gro86] L. K. Grover, "A New Simulated Annealing Algorithm for Standard Cell Placement," *Proc. IEEE International Conf. on Computer-Aided Design*, 1986, pp. 378-380.

[MitRom86] D. Mitra, R. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and Finite-Time Behavior of Simulated Annealing," *Advances in Applied Probability*, Vol. 18, No. 3, pp. 747-771, 1986.

[KleSig91] J. M. Kleinhans, G. Sigl, F. M. Johannes and K. J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Trans. Computer-Aided Design*, Vol. 10, No. 3, March 1991, pp. 356-365.

[SigDol91] G. Sigl, K. Doll and F. M Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?" *28th ACM/IEEE Design Automation Conference*, 1991, pp. 427-432.

[DolJoh91] K. Doll, F. M. Johannes and G. Sigl, "DOMINO: Deterministic Placement Improvement with Hill-Climbing Capabilities," *Proc. VLSI*, 1991, pp. 91-100.

[DolJoh92] K. Doll, F. M. Johannes, and G. Sigl, "Accurate Net Models for Placement Improvement by Network Flow Methods," *Proc. IEEE International Conf. on Computer-Aided Design*, 1992, pp. 594-597.

[CheKuh84] C.-K. Cheng and E. S. Kuh, "Module Placement Based on Resistive Network Optimization," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 3, No. 3July 1984, pp. 218-225.

[HuaKah97] D. J.-H. Huang and A. B. Kahng, "Partitioning-Based Standard-Cell Global Placement with and Exact Objective," *Proc. ISPD*, April 1997, pp. 18-25

[KinBan89] R. M. King and P. Banerjee, "ESP: Placement by Simulated Evolution," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 8 No. 3, March 1989, pp. 245-256.

[EtaVan98] H. A. Etawil and A. Vannelli, "Utility Function Based Hybrid Algorithm for Channel Routing," *Proc. IEEE International Conf. on Computer-Aided Design* 1998, pp. 258-261

[Ken97] A. Kennings, "A Deterministic Iterative Improvement Technique," Ph.D. Thesis, University of Waterloo, Waterloo, Ontario, Canada, 1997.

[EtaAre99] H. A. Etawil, S. Areibi, and A. Vannelli, "Attractor-Repeller Approach for VLSI Global Placement," *Proc. IEEE International Conf. on Computer-Aided Design*, 1999.

[SecLee87] C. Sechen and K-W Lee, "An Improvement Simulated Annealing Algorithm for Row-Based Placement," *Proc. IEEE International Conf. on Computer-Aided Design*, 1987, pp. 478-481.

[SecSan85] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE J. of Solid-State Circuits*, Vol. 20, No. 2, 1985, pp. 432-439.

[Koz91] K. Kozminski, "Benchmarks for Layout Synthesis - Evolution and Current Status," *28th ACM/IEEE Design Automation Conference*, 1991, pp. 265-270.

[SecSan86] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf 3.2: A New Standard Cell Placement and Global Routing Package," *Proc. Design Automation Conference*, 1986, pp. 432-439.

[HanWol76] M. Hanan, P. K. Wolff, Sr., and B. J. Agule, "Some Experimental Results on Placement Techniques," *Proc. Design Automation Conference*, 1976, pp. 214-224.

[Are95] S. Areibi, "Towards Optimal Circuit Layout Using Advanced Search Techniques," Ph.D. Thesis, University of Waterloo, Waterloo, Ontario, 1995.

[WesEsh93] N. H. E. Weste and K. Eshraghian, "Principles of CMOS VLSI Design: A Systems Perspective," Addison-Wesley, New York, 1993.

[Alp94] C. J. Alpert and A. B. Kahng, "A General Framework for Vertex Orderings, With Applications to Netlist Clustering," *Proc. IEEE International Conf. on Computer-Aided Design*, 1994.

[Kar97] George Karypis et. al., "Multilevel Hypergraph Partitioning: Applications in VLSI Domain", *Proc. Design Automation Conference*, 1997.

[Kar99] George Karypis et. al., "The hMetis Hypergraph Partitioning Package," http://www.cs.umn.edu/ karypis.

[SchUlr72] D. M. Schuler and E. Ulrich, "Clustering and Linear Placement," *Proc. Design Automation Workshop*, 1972, pp. 50-56.

[MalGro89] Sivanarayana Mallela and Lov K. Grover, "Clustering based Simulated Annealing for Standard Cell Placement", *Proc. Design Automation Conference*, 1989, pp. 312-317.

[Eta99] H. Etawil, "Convex Optimization and Utility Theory: New Trends in VLSI Circuit Layout," Ph.D. Thesis, University of Waterloo, Waterloo, Ontario, 1999.

[YehChe92] C. W. Yeh, C. K. Cheng, and T. T. Lin, "A Probabilistic Muticommodity-Flow Solution to Circuit Clustering Problems," *Proc. IEEE International Conf. on Computer-Aided Design*, 1992, pp. 428-431.

[ConHag91] J. Cong, L. Hagen, and A. Kahng, "Random Walks for Circuit Clustering," *Proc. Intl. Conf. on ASIC*, June 1991, pp. 14.2.1-14.2.4.

[KahKan92] L. Hagen and A. B. Kahng, "A New Approach to Effective Circuit Clustering," *Proc. IEEE International Conf. on Computer-Aided Design*, 1992, pp. 422-427.

[GarPro90] J. Garbers, H. J. Promel, and A. Steger, "Finding Clusters in VLSI Circuits," *Proc. IEEE International Conf. on Computer-Aided Design*, 1990, pp. 520-523.

[HagKah92] L. Hagen and A. B. Kahng, "New Spectral Methods for Ration Cut Partitioning and Clustering," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.11, No.9, Sept. 1992, pp. 1074-1085.

[AlpKah96] C. J. Alpert and A. B. Kahng, "Simple Eigenvector-Based Circuit Clustering Can Be Effective," *Proc. IEEE Intl. Symp. on Circuits and Systems*, May 1996, pp. IV/683-686.