

Copyright © 1993, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**COMBINED HIERARCHICAL APPROACHES
TO INTEGRATED CIRCUIT LAYOUT BASED
ON A COMMON DATA MODEL**

by

Brian Douglas Ngai Lee

Memorandum No. UCB/ERL M93/81

22 November 1993

**COMBINED HIERARCHICAL APPROACHES
TO INTEGRATED CIRCUIT LAYOUT BASED
ON A COMMON DATA MODEL**

Copyright © 1993

by

Brian Douglas Ngai Lee

Memorandum No. UCB/ERL M93/81

22 November 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

**COMBINED HIERARCHICAL APPROACHES
TO INTEGRATED CIRCUIT LAYOUT BASED
ON A COMMON DATA MODEL**

Copyright © 1993

by

Brian Douglas Ngai Lee

Memorandum No. UCB/ERL M93/81

22 November 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

*To Donna
for her love and inspiration*

Abstract**Combined Hierarchical Approaches to Integrated Circuit Layout Based on a
Common Data Model**

by


Brian Douglas Ngai Lee

Doctor of Philosophy in
Electrical Engineering and Computer Science
University of California at Berkeley
Professor A. Richard Newton, Chair

The integrated circuit layout problem is the specification of fabrication patterns that implement a given circuit description subject to manufacturing and performance constraints. The high complexity of integrated circuits has made it necessary to decompose the layout problem into a set of subproblems, and the typical layout process usually solves a sequence of subproblems corresponding to the steps of floor planning, placement, channel definition, global routing, detailed routing, and compaction. Each subproblem depends on the solutions of the previous subproblems and iteration over the sequence of problems may be required to find a feasible layout solution. Unfortunately, iterative improvement of layout solutions is very difficult because a side effect of the partitioning is that feedback between subproblems is hard and not well understood.

A merged hierarchical approach to layout is presented that provides a means of investigating the relationship between placement and routing to improve current layout methods. A common data model is used to integrate layout phases and to simplify and enhance information flow within the layout process. The data model induces a structure to the layout process for experimenting with feedback and information management.

A layout system based on a 2×2 grid-graph abstraction that combines placement and global routing has been implemented based on this paradigm. Examples from sea-of-gates designs are used for benchmark comparisons.



Prof. A. Richard Newton
Dissertation Committee Chairman

*To Donna
for her love and inspiration*

List of Tables

1.1	Example characteristics	13
3.1	Relative edge weights by abstract net type and edge	37
3.2	Comparison of linear versus nonlinear objective function	45
3.3	Statistics on the number of non-integer programs solved during the branch-and-bound solution of integer programs	45
3.4	Effects of applying a simple re-route procedure	51
4.1	Total net length comparisons between the separate phase 2×2 algorithms and TimberWolf 6.1	59
4.2	Area and aspect ratio comparisons between the separate phase 2×2 algorithms and TimberWolf 6.1	60
5.1	Edge capacity violations as a function of the number of levels of cell assignment performed ahead of route assignment	68
5.2	Edge capacity violations and total net length for the immediate and gradual methods of pseudo-pin assignment	70
6.1	Merged placement and routing versus separate phases	81
6.2	Coupled placement and routing versus separate phases	86
6.3	Hybrid placement and routing versus separate phases	92

Acknowledgements

First, I would like to thank Professor Richard Newton, my research advisor, for his generous and constant support and patience throughout my graduate school years. His vision and ideas maintained my interest, provided encouragement, and spurred my creativity.

I am grateful to Professor Carlo Séquin and Professor Charles Stone for reading this dissertation and for being on my committee.

I also thank Professors Robert Brayton, Richard Newton, Donald Pederson, and Alberto Sangiovanni-Vincentelli for creating the enjoyable and stimulating research environment called the Berkeley CAD group. I especially thank Professor Sangiovanni-Vincentelli for serving as my qualifying exam committee chairman and Professor Pederson for starting me, as a freshman, on the path that eventually led to this degree.

Many other members of the Berkeley CAD group, both past and present, contributed beneficially to my graduate school experience. To mention but a few, I thank Mark Beardslee, Jeff Burns, Andrea Casotto, Wayne Christopher, David Harrison, Chuck Kring, Tom Laidig, Peter Moore, Tom Quarles, Ellen Sentovich, Greg Sorkin, and Rick Spickelmier for all their time and help and for generally making my stay here fun and worthwhile. I am especially grateful to Ellen Sentovich and Mark Beardslee for their support, advice, and friendship these past few years.

This research was sponsored in part by the Defense Advanced Research Projects Agency, the Digital Equipment Corporation, and the Semiconductor Research Corporation. Their support is gratefully acknowledged.

Thanks also go to my friends in the Berkeley Hang Gliding Club for aiding and abetting my much needed escapes from reality to freedom in the air. I wish them all good lift and safe flights.

I am also grateful to the extended "Pajaro Dunes Gang" for their support, encouragement, and gentle nagging.

A special thanks goes to my parents and family for their constant love and faith. They put up with much and never gave up on me.

Finally, and with much love, I thank Donna for being there at the end to help me finish.

List of Figures

1.1	Gate array design structure	5
1.2	Standard cell design structure	6
1.3	Macro-cell design structure	7
1.4	Sea-of-gates design structure	8
1.5	Typical layout system design flow	10
2.1	Classification of chip level integrated placement and routing methods	16
3.1	2×2 data model	26
3.2	Problem hierarchy created by a hierarchical decomposition	28
3.3	Example decomposition step	30
3.4	Basic quadrisection algorithm pass	35
3.5	Possible routes on the 2×2 abstraction	38
3.6	2×2 net types and configurations	40
3.7	Uneven route distribution	43
3.8	Immediate pseudo-pin assignment	49
3.9	Gradual pseudo-pin assignment	50
4.1	Placement constraints for terminal propagation	54
4.2	Subproblem abstraction on a placement grid	56
5.1	Subproblem abstraction sequence for solving a routing level immediately after the corresponding placement level	64
5.2	Subproblem abstraction sequence for solving one more level of placement before routing	65
5.3	Subproblem abstraction sequence for solving all levels of placement before routing	66
6.1	Merged approach subproblem abstraction sequence	78
6.2	Routing and pseudo-pin assignment influence on placement	80
6.3	Effect of re-placement on net connected components	82
6.4	Initial coupled approach subproblem abstraction sequence	84
6.5	Remaining coupled approach subproblem abstraction sequence	85
6.6	Initial hybrid approach subproblem abstraction sequence	88

6.7 Remaining hybrid approach subproblem abstraction sequence 89

Contents

List of Figures	vi
List of Tables	viii
1 Introduction	1
1.1 Integrated Circuit Layout Problem Definition	2
1.2 Design Styles	4
1.3 Typical Layout Process	9
1.4 Problems with the Typical Layout Process	12
1.5 Goals	13
1.6 Benchmark Examples	13
2 Background	15
2.1 Coupled Placement and Routing	17
2.2 Merged Placement and Routing	18
2.3 Unified Placement and Routing	22
2.4 Summary	22
3 Algorithms	24
3.1 Hierarchical Decomposition	25
3.1.1 Common Data Model	25
3.1.2 Problem Reduction	26
3.2 Subproblem Solution	32
3.2.1 Cell Assignment	33
3.2.2 Route Assignment	38
3.3 Pseudo-pin Assignment	46
3.4 Rip-up and Re-route	48
4 Separate Placement and Routing	52
4.1 Placement	52
4.2 Routing	55
4.3 Results	57

5 Placement and Routing Communication	61
5.1 Routing	61
5.1.1 Placement Resolution	62
5.1.2 Routing Decision Resolution	69
5.2 Placement	71
6 Combining Placement and Routing	73
6.1 Unified Approach	74
6.2 Merged Approach	76
6.3 Coupled Approach	83
6.4 Hybrid Approach	87
7 Conclusion	94
Bibliography	97

Chapter 1

Introduction

The physical design of integrated circuits requires the transformation of a circuit description into *layout*, the set of geometric patterns needed to generate the fabrication masks used in very large scale integration (VLSI) manufacturing processes. This layout generation step is a difficult stage in the design process of an integrated circuit and the effectiveness of layout generation systems affects both the cost and performance of the final product. The research presented in this dissertation is intended to indicate layout paradigm modifications that will produce higher quality layout solutions.

The layout problem is to produce layout that implements a given circuit description subject to manufacturing and performance constraints. Current circuits may contain millions of devices, resulting in complex and difficult layout problems. Today's market pressures make continual improvement in layout solution quality an imperative for current layout systems.

The physical design stage of an integrated circuit has an important impact on the performance and cost of the final product. As the last step in the circuit design process, layout generation is the closest stage to the manufacturing process and its function as the major link between design and production makes it a crucial determinant of ultimate circuit performance relative to the underlying fabrication technology. The effectiveness of the layout solution process directly affects design cycle time and thus total design cost. Layout solution quality directly affects yield and thus manufacturing cost. Both design cost and manufacturing cost are major components of the final product cost.

Unfortunately, the combinatorial nature of the layout problem makes the generation of layout a difficult task. Exact solution methods are prohibitively expensive except

for trivial cases, so layout algorithms tend to rely on heuristics and approximations to find reasonable solutions. The typical layout system solves a layout problem by decomposition into an ordered set of simpler subproblems and then sequential solution of the subproblems. These subproblems are referred to as *layout phases* and may include stages called floor planning, placement, channel definition, global routing, detailed routing, and compaction. Because of the difficulty and complexity of generating layout, this decomposition is often considered essential and is usually assumed implicitly in any discussion of layout algorithms. Despite the problem reduction, the “simpler” subproblems are actually still difficult combinatorial optimization problems. The phases are easier only in the sense that heuristic and approximate methods have been found which produce acceptable solutions for them.

The difficulty of the individual phases has focused layout research on solutions for each phase. An unfortunate side effect of this emphasis is that the interaction and dependencies between phases have been neglected. Information flow and feedback between phases is often nonexistent or ad hoc. Ideally, a layout algorithm would solve the layout problem as a single problem, but since some form of problem decomposition is required to make the problem tractable, enhancing communication between subproblem solution processes is critical for improving the effectiveness of the whole approach. The goal of this research is to obtain better layout solutions by modifying the standard layout decomposition paradigm to allow more structured and effective information management. In particular, integration of placement and routing phases and mechanisms for feedback are investigated.

In the remainder of this chapter, more introductory material is presented on the layout problem, the characteristics and problems of typical layout processes, and the goals for improving these processes. A discussion of previous work in combining layout phases is given in Chapter 2. The major algorithms used are described in Chapter 3 and reference results are presented in Chapter 4. Placement and routing interaction is discussed and analyzed in Chapter 5 and experiments in integrating placement and routing based on a common 2×2 data model are detailed in Chapter 6.

1.1 Integrated Circuit Layout Problem Definition

Current VLSI manufacturing processes create circuits by building layers of different materials. The patterns and combinations of these layers create electrical devices and the interconnections between these devices. Fabrication requires a set of geometric

patterns for each layer, and the patterns are used to create the templates or masks for processing each layer. The layout problem is to transform a circuit description into the appropriate geometric patterns for generation of fabrication masks. For the typical layout system, this transformation requires taking a circuit description and determining circuit element positions and wire positions that implement the circuit subject to manufacturing and performance constraints.

The *net list* is a common form of circuit description. It consists of a set of interconnections between circuit elements. The interconnections specify which elements must be electrically connected to each other. Each interconnection is called a *net* and represents an electrically equivalent point in the circuit. Hence, circuit descriptions presented as collections of interconnections are called "net lists". Circuit elements may be single devices like transistors or collections of devices such as logic gates or other circuit building blocks. These elements are often referred to as *cells* or *modules*. The term "net" is also used to refer to the wire or wire patterns that implement an interconnection. Wires connect to cells at specific locations called *pins* or *terminals*, so a net may be associated with a set of cells that require interconnection or with the corresponding set of pins.

Cells or circuit elements are often used multiple times within a design and across designs, so the sets of geometric patterns that implement particular circuit elements are usually predefined. However, the patterns that implement the wiring corresponding to each net are not predefined because in general these interconnections are unique to each circuit. Fortunately, the interconnection patterns can often be derived from specification of simpler, abstract wire positions. Thus, solving the layout problem involves assigning cell positions and determining abstract wire positions. The process of assigning cell positions is called *placement* and the process of assigning wire positions is called *routing*. This information is usually sufficient to generate the set of geometric patterns required for generation of the actual fabrication templates.

The solution of a layout problem requires that the final geometric patterns satisfy certain problem specifications. These specifications may be grouped into manufacturing constraints and performance constraints. Proper fabrication requires that certain requirements be met on the size, shape, and relative positions of the geometric patterns both within layers and between layers. These requirements are referred to as *design rules*. Typically, these rules are translated into constraints on cell and wire positions such that a feasible set of positions may always be transformed into a set of design-rule correct geometry. Per-

formance constraints are usually specified with respect to the fabricated circuit. The most common example is timing requirements on the circuit inputs and outputs. Unfortunately, it is often difficult to relate constraints of this type to the specification of layer patterns. Often, accurate testing of constraint satisfaction is only possible after complete specification of the layout. This kind of dependence makes finding layout solutions difficult.

Strictly speaking, a feasible layout solution must meet all design constraints. However, because of the simplifications and approximations made by many layout systems, a feasible layout solution often refers only to a solution that may be correctly manufactured. In these cases, the performance constraints have been ignored and the usage is usually clear from context.

1.2 Design Styles

Gate array, standard cell, and macro-cell layout are the most common design styles implemented by automatic layout systems. A brief review of these styles follows to provide context for the subsequent discussion on layout phases. The examples and layout system in this dissertation are based on a specific type of gate array layout called sea-of-gates.

Gate array designs are also referred to as master-slice designs or uncommitted logic arrays. Each design is based on a two dimensional array of transistor blocks. This structure is depicted in Figure 1.1. The array is completely prefabricated except for the interconnection layers. Each transistor block consists of a solid group or array of transistors. Circuit elements or functions are implemented by customization of interconnect patterns within each local transistor group. The interconnect patterns for common circuit functions are stored as templates in cell libraries. Connections between circuit elements are made in the predefined wiring areas or *channels* left open between transistor blocks. Though the use of partially fabricated standard arrays is advantageous for manufacturing, it creates difficult constraints on the layout problem. In particular, because of the predefined and fixed areas for routing, the complete connection of circuit elements may not be possible for a given assignment of elements to transistor blocks.

Standard cell designs are characterized by rows of cells separated by wiring or routing channels. This structure is illustrated in Figure 1.2. Cells are custom-designed to implement circuit functions on the level of simple logic gates, flip-flops, and latches, and are often constrained to have uniform height to simplify wiring between rows and power

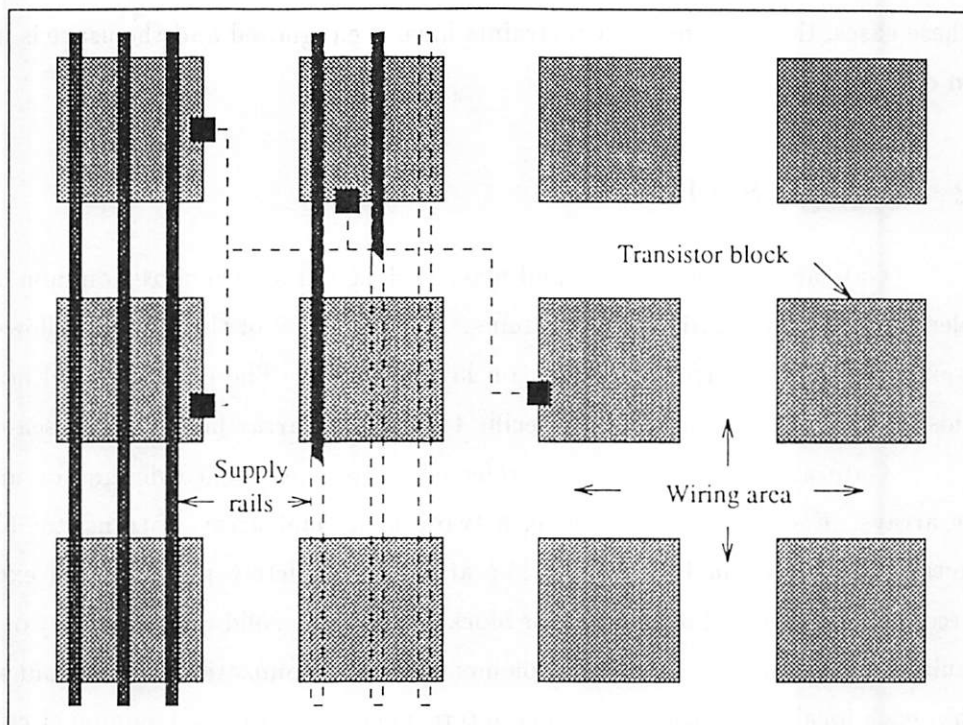


Figure 1.1: Gate array design structure

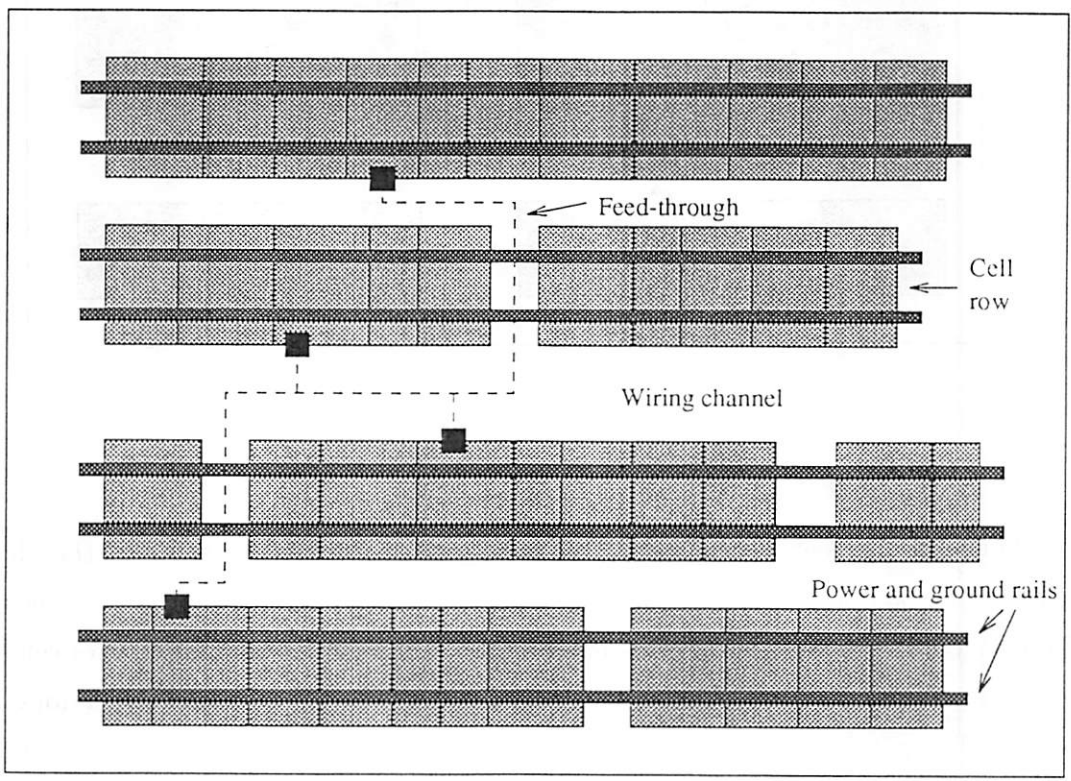


Figure 1.2: Standard cell design structure

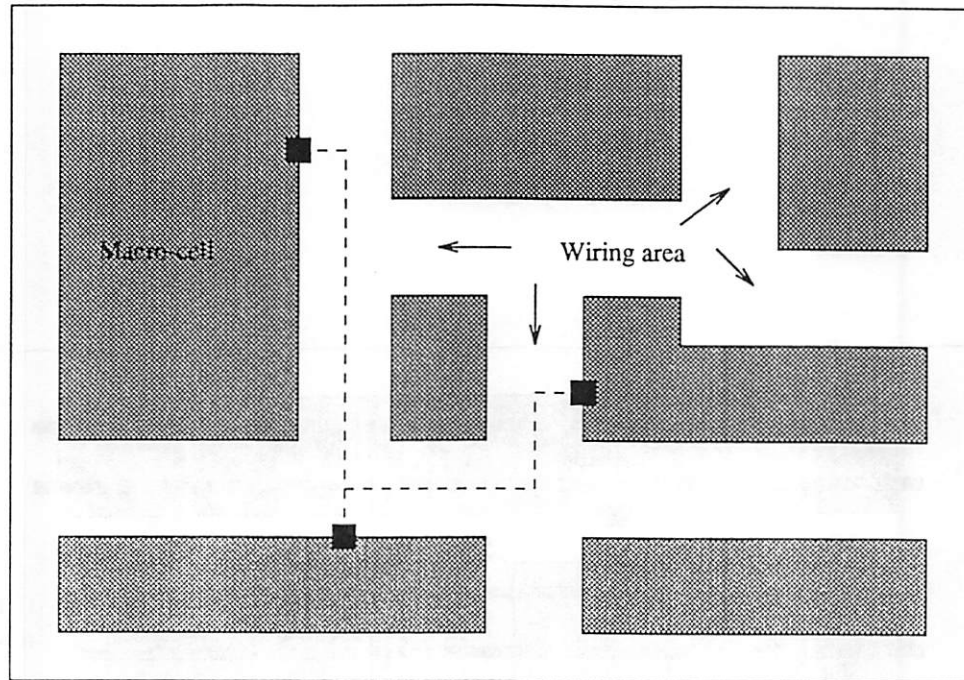


Figure 1.3: Macro-cell design structure

distribution along rows. Typically, all wiring between cells is confined to the channels between rows and any connections that cross rows must use reserved spaces between cells. These spaces are called *feed-throughs* and are often modeled as a special type of cell. More sophisticated systems are capable of wiring over or through cells to make connections across rows. Unlike gate arrays, standard cell designs specify design-dependent geometry on all mask layers. This characteristic permits the number of feed-throughs and the amount of wiring area between rows to be adjusted as necessary and guarantees that feasible wire assignments for all nets can always be generated.

Macro-cell or building-block designs differ from standard cell designs in that each cell implements a much larger amount of functionality and the cell topology is less constrained. While each standard cell typically implements a simple logic function, each macro-cell usually implements a major circuit function. In fact, the predefined patterns of the macro-cells are occasionally generated using standard cell design. Macro-cells may vary widely in size and shape and may be positioned in any non-overlapping configuration. Figure 1.3 shows an example macro-cell design structure. Similar to standard cells, the

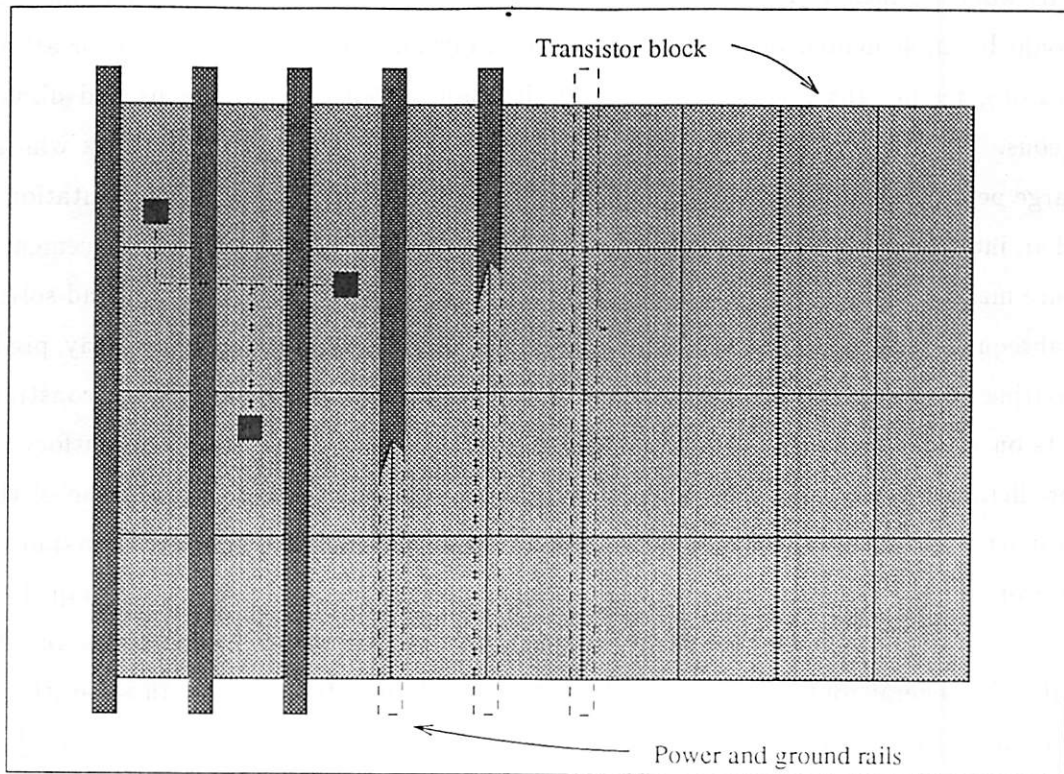


Figure 1.4: Sea-of-gates design structure

patterns for inter-cell wiring connections are specified in the regions between and around cells. These wiring spaces can be adjusted as required to allow complete specification of the layout.

Sea-of-gates or “channel-less” gate-array designs are gate arrays without any reserved wiring area between transistor blocks. This structure is illustrated in Figure 1.4. Connection patterns within cells are usually sparse and inter-cell wiring patterns are specified both through cells and over cells. In the desired interconnection regions, extra area for wiring can be obtained by not utilizing some of the transistor blocks for circuit element implementation. A goal of sea-of-gates design systems is to use as few transistor blocks as possible for this extra interconnection area, resulting in a maximal amount of functionality implemented for a given array size. Relative cell positions are unconstrained except for the grid restrictions which force spacing between cells to occur in multiples of transistor blocks. Of course, simplifying restrictions could be added to force the use of cell rows, but

this would artificially limit the number of cells and thus, the amount of functionality that could be implemented on a sea-of-gates array, invalidating a major reason for attempting sea-of-gates designs. The unconstrained situation is both advantageous and disadvantageous. The freedom in choosing cell positions may allow very good solutions which use a large percentage of the available transistor blocks for circuit element implementation rather than interconnection implementation, but this same lack of enforced cell placement structure makes finding feasible solutions more difficult. Allocating routing area and solving the subsequent routing problems is especially difficult. Unlike a standard gate array, predefined routing areas do not exist and unlike a standard cell design, no simplifying constraint exists on the positioning of cells to make the formation of routing areas straightforward and predictable. Also, moving cells apart may incur a large area penalty because of the grid restrictions and often may require a complete re-placement of the design. Maximizing the use of transistor blocks for circuit cell implementation leads to arbitrarily shaped routing areas and requires heavy use of interconnection area over and through cells, resulting in difficult, general routing problems. As in the standard gate-array design style, there is no guarantee that it will be possible to completely connect a particular assignment of cells to blocks.

1.3 Typical Layout Process

Typical layout systems solve the layout problem in several phases. The phases are sequential and dependent. The common phases and their relationship in a typical layout design flow is shown in Figure 1.5.

Sometimes, circuit cells do not have their corresponding geometric patterns completely specified. This occurs in macro-cell design where a cell is a major circuit function consisting of multiple sub-circuits or devices. The functionality of the cell is known, but not its layout. The **floor planning** phase handles these kinds of specifications. Given bounds on cell areas, the floor planning phase determines relative cell positions and if necessary shapes for cells. The process allocates routing area both between cells and within cells and determines pin locations on the newly assigned cell shapes. Subsequent phases solve the layout problems delineated by the now defined macro-cell regions. For this phase, the generation of a certain chip aspect ratio is a typical constraint and the minimization of total chip area is a common objective function.

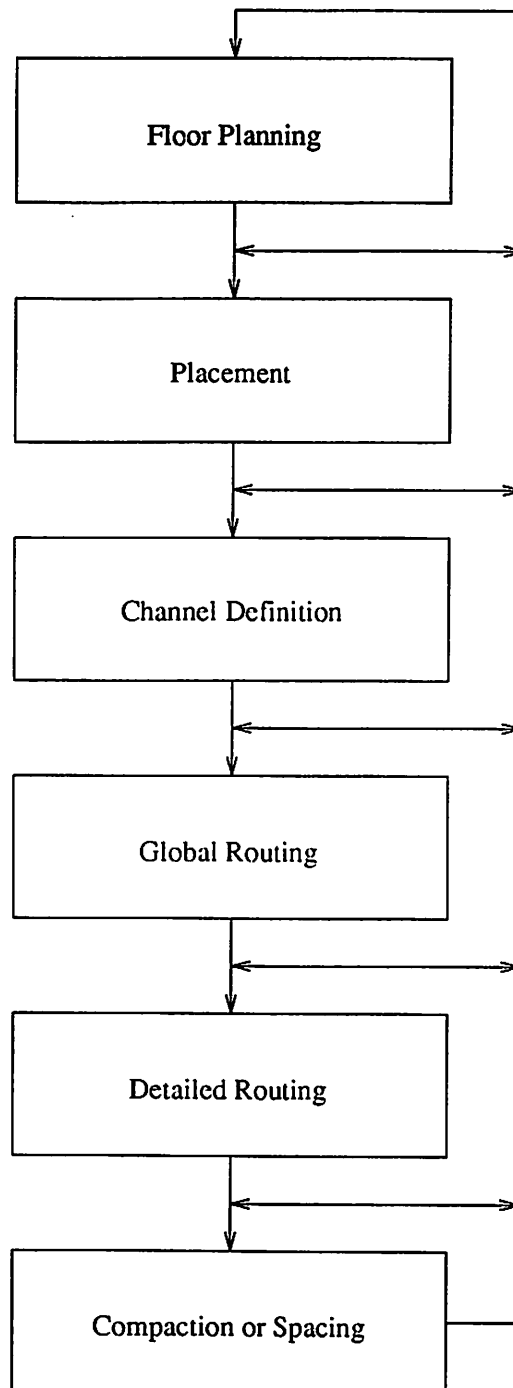


Figure 1.5: Typical layout system design flow

The **placement** phase specifies cell positions on the chip or within regions specified by the floor planning stage. Cells are not allowed to overlap and constraints on relative cell positions depend on the design style being implemented. For example, gate array styles require cell assignments on specific grid locations while other styles are less constrained. Typical placement goals are minimization of area, total net length, or both while producing a feasible placement. The feasibility condition ultimately refers to the feasibility of the subsequent and resultant routing problem. Unfortunately, this condition is often impossible to check without actually performing the routing step.

The **channel definition** step defines the wiring regions on the chip. Its name originates in design styles where wires are routed in the "channels" between cell rows. More specifically, the term channel refers to a routing problem with two parallel boundaries. Pins are specified at positions along these boundaries. Nets may be assigned to enter or exit from the open sides of the channel, but neither the position nor relative order of these net crossings are specified. The distance between the parallel boundaries of a channel routing problem is allowed to increase to provide more area to complete the interconnections. In standard cell design, the channel definition occurs implicitly as a result of the routing. In macro-cell layout, this step determines explicitly the sizes and positions of the routing areas between cells. It is possible to constrain macro-cell placements so that the regions between cells can always be partitioned into channels. In systems without this constraint, the channel definition stage may also create *switchboxes*. The classic switchbox is a rectangular routing region with pins specified on all borders. It can be thought of as a channel routing problem where the distance between channel boundaries is fixed and where pins are specified at fixed positions along the open sides of the channel. In gate-array styles, routing also occurs over and through cells and it is more appropriate to think of the channel definition step as routing area definition.

The routing phases perform the actual assignment of inter-cell wire positions. This is commonly performed in two phases. The first phase, **global** or **loose routing**, assigns nets to particular routing channels or routing areas. After this step, the path of each wire is known with respect to the areas defined during the channel definition phase. The second phase, **detailed routing**, assigns specific positions for the wires within the regions assigned during global routing.

After routing, the circuit layout is basically complete. However, some systems employ a final **compaction** or **spacing** step where all layout geometries are squeezed

together or re-spaced to generate a final layout of minimal area that still meets design rule constraints.

1.4 Problems with the Typical Layout Process

The fundamental problem of the typical layout process is that it doesn't directly solve the actual, desired layout problem. Instead of solving a single optimization problem subject to the multitude of possible layout constraints, typical layout systems solve a sequence of simpler optimization problems subject to a subset of the possible constraints. Historically, this decomposition into layout phases and relaxation or omission of constraints has been necessary in order to generate any layout solution. Unfortunately, this strategy only solves an approximation of the desired problem and can not guarantee that a feasible solution to the actual layout problem will be found.

Problems arise in the typical layout process from the manner and method in which the decomposition occurs rather than from the decomposition into layout phases itself. As in any sequential decomposition, the subproblem solved at each phase depends on the solutions of the previous phases and iteration over the sequence of phases may be required to find a feasible layout solution. Unfortunately, iterative improvement of layout solutions obtained in this manner is very difficult because a side effect of the decomposition is that feedback between subproblems is hard and not well understood. Each phase solves only a portion of the whole layout problem. Each of these portions may be thought of as an abstraction or approximation of the desired problem. As a result, each phase has its own data model of the layout problem that is different from the other phases. For successful iterative improvement, each phase needs some global information regarding decisions made in previous phases. Typical layout systems have many barriers to the effective flow of this information. Often, each phase is performed by a separate computer program which only understands its particular data model. Any information from other phases must be translated or interpreted relative to the current problem abstraction. In general, this translation or interpretation is not well understood, and even in specific cases where the interpretation is clear, differences between the data models may lead to information loss. Furthermore, the separate programs tend to have simple interfaces which prohibit the exchange of the desired global information. These factors inhibit and discourage the iterative feedback required to find feasible layout solutions.

Example	Cell Count	Pad Count	Net Count	Cell Area (λ^2)
C880	257	86	317	1989951
C1355	292	73	333	2260956
misex3c	295	28	309	2284185
duke2	299	51	321	2315157
C1908	326	58	359	2524218
misex3	344	28	358	2663592
C3540	784	72	834	6070512
C5315	1073	301	1251	8308239
C7552	1365	315	1572	10569195
C6288	2385	64	2417	18467055

Table 1.1: Example characteristics

1.5 Goals

The main goal of this research is to find modifications to the standard layout paradigm that will produce better layout solutions. This goal is pursued by examining placement and routing interaction and information exchange requirements to understand the relationship between the two phases better, analyzing different placement and routing integration and feedback approaches, and evaluating the use of a common data model to improve communication and reduce information loss between phases due to translation or interpretation inefficiencies. The investigations into these areas are designed to indicate methods for improving placement and routing through closer coupling or merging of the two phases. The experiments are based on a layout system that hierarchically decomposes the layout problem using a common data model. The resulting hierarchy provides a convenient structure for observing placement and routing interaction and allows comparisons of different levels of placement and routing integration from separate phases to concurrent solution. The common data model provides a common reference frame for observing placement and routing communication and for experimenting with feedback mechanisms.

1.6 Benchmark Examples

The benchmark examples used in this research are characterized briefly in Table 1.1. The examples are derived from the MCNC and ISCAS logic synthesis benchmark

circuits [45]. Circuit functions include error correction logic, ALUs and their control logic, a 16-bit multiplier, and generic control logic. Each circuit was processed using `sis` [33] and mapped into the Mariner sea-of-gates library [23]. `Sis` is an algorithmic sequential circuit optimization program. It was used both to optimize the examples using its standard set of operations and to map the optimized circuits into the sea-of-gates library. The resulting circuits have their cell, pad, and net counts listed in the table. The cells themselves range in complexity from a simple NAND gate with four transistors to an AND-OR-INVERT gate with eight transistors. There are two interconnection layers; the primary vertical interconnection layer is Metal 1 and the primary horizontal interconnection layer is Metal 2. The library cells used by these examples have from four to six feed-throughs for through-the-cell routing and from two to ten free tracks for over-the-cell routing.

Chapter 2

Background

Layout algorithms have been studied extensively for many years. Continued interest in this area emphasizes the importance of layout generation to product design. In addition to handling increased circuit sizes and adjusting to rapidly changing manufacturing technologies and design styles, current layout systems must also produce better layout solutions. The discussion in this chapter is limited to previous work in combining placement and routing at the chip level. More general discussion and surveys on placement and routing algorithms can be found in [5, 37, 34].

A major reason for the interest in combining placement and routing is the promise of finding better layout solutions. In the past, research emphasis was concentrated on the separate phases. This was a manifestation of the complexity of the layout problem and the lack of understanding about the relationship between phases. Concentrating on each phase produced results more effectively than tackling the combined problem. Now, as computer power increases, the complexity of some methods of combining phases is becoming tractable, and as the benefits of continued optimization of individual phases decrease, the potential gain from investigating integrated processes increases.

Efforts to combine placement and routing at the chip level can be categorized into three broad classes that span a continuum of algorithm integration possibilities as depicted in Figure 2.1. *Coupled* approaches execute separate placement and routing phases and then feed back between the two phases. During each phase, a complete solution is found. For these classifications, a complete placement solution is a specification of all cell positions relative to each other and a complete routing solution is an assignment of all nets to global routing configurations. *Merged* approaches generate complete placement and routing

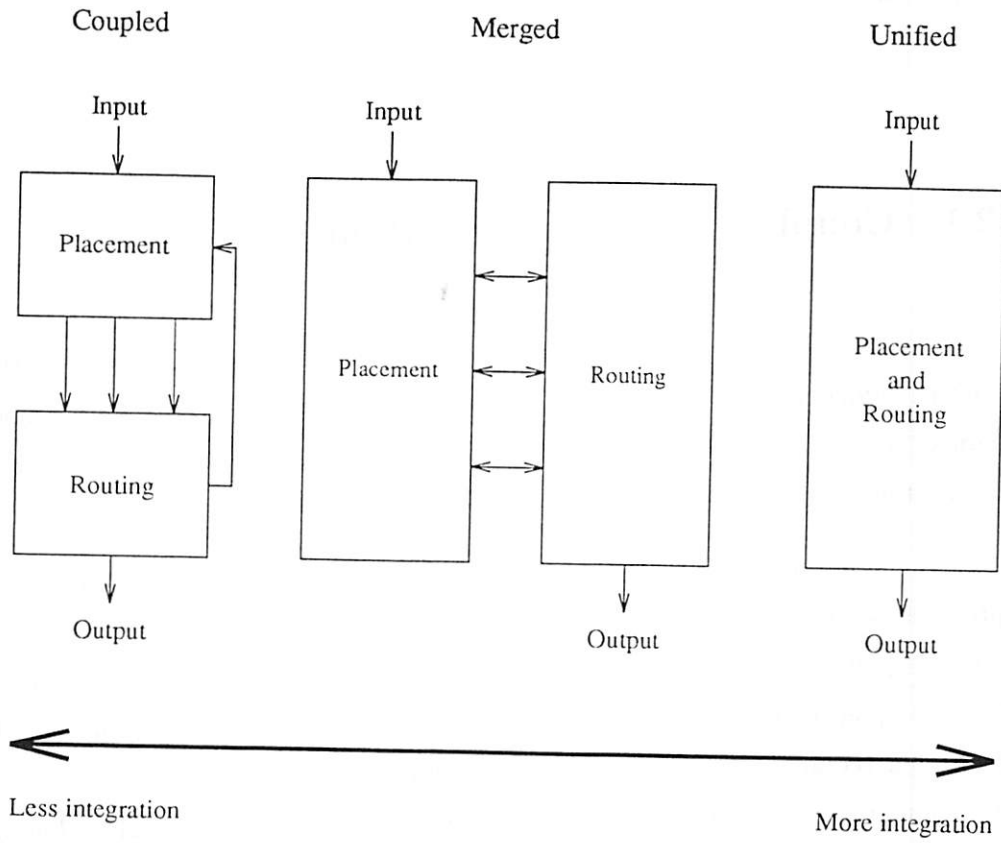


Figure 2.1: Classification of chip level integrated placement and routing methods

solutions concurrently. These methods are characterized by a more intense exchange of information than in the coupled approaches. The level of feedback information is comparable in magnitude to the forward flow of information. *Unified* approaches determine placement and routing together as a single problem. Coupled methods are the least integrated and unified approaches are the most integrated. Recently, Adams [1] has worked on integrating layout phases for macro-cell design. No attempt is made here to classify his or other cell-level approaches.

2.1 Coupled Placement and Routing

Coupled placement and routing methods consist mainly of systems that feed back routing information to re-space placements. Typically, the relative cell topology is maintained in a graph or other global data structure and routing information is fed back to adjust the spacing between cells.

Soupkup and Royle [38] presented a prototype building-block layout system that provides some automatic feedback between routing and placement. This system uses a min-cut placement algorithm based on polar graph representations [21] to generate the basic cell placement topology. The routing regions between cells are represented as possibly zero-sized rectangles and edges for these rectangles are introduced into the polar graphs. Subsequent routing provides size requirements for the routing rectangles. Though the basic layout topology is not changed, size information is fed back to the placement to effect cell re-spacing that accommodates routing requirements.

The Orca sea-of-gates system [17] uses a min-cut placement algorithm [19, 15] to form a cut tree of relative cell topology. Locations and shapes of cell groups within this cut tree are determined by a "shape-adding" approach [39]. A simplified global router is used to determine space requirements in over-congested regions. Spacing information is fed back to the placement by modifying the shape function of the appropriate cell group. The "shape-adding" algorithm and the global routing are iterated until the design can be routed successfully at the detailed level.

2.2 Merged Placement and Routing

Most work on integrating placement and routing can be classified as merged approaches. These are primarily hierarchical methods. Two exceptions are the systems of Shragowitz, Lee, and Sahni [36] and Tzeng [43]. For the merged approaches, the term *hierarchical* refers to a structuring of the process and data in terms of a hierarchy or tree of decisions or subproblems. Placement and routing can be combined in these methods by solving both placement and routing problems at each step or level of the hierarchy.

Burstein, Hong, and Pelavin [9] presented a merged placement and global routing approach for gate array VLSI layout. The process is structured according to a bi-partitioning hierarchy. Starting with a single super-grid corresponding to the entire layout region, each set of subdivisions creates a new level of super-grids until the final level corresponds to the gate array grid. All grid cells within a level are cut in the same direction, either horizontal or vertical, and directions are alternated between levels. When a super-grid cell is divided, the cells previously assigned to that super-grid must be partitioned into two parts corresponding to the next level grid cells. An initial partition is formed by heuristically selecting one cell at a time to be added to one partition block or the other. Then, the number of nets crossing the partition boundary is compared to the number of available wiring tracks and a pairwise interchange routine may be used to reduce the number of crossing nets. On each level, each net is assigned a global route relative to the current super-grid. The global routing of a net is initially restricted to a region projected from the net's assigned route path on the previous level. A route path is defined by a set of grid cells and when it is projected onto the next level, it covers twice as many cells. This new set of cells can be subdivided into rectangles of width no greater than 2. Each of these rectangles is routed as a $2 \times n$ grid using Burstein and Pelavin's dynamic programming technique [10]. If a route can not be found within this restricted set of cells, then maze routing is used across all the cells at the current level to find a path.

Szepieniec [42] proposed a method of integrating placement and global routing for slicing structures. The approach begins with a given slicing structure. In this slicing structure, the number of slices, the relative sizes of slices, and the distribution of circuit elements across slices are fixed. Within each slice, the placement of subslices is solved as a linear placement problem. The objective of each linear placement problem is to minimize the geometric span of all nets. Hierarchical global routing is performed top-down along

with the slice placement. The routing problem for each slice is formulated after subslice placement by determining net entry positions along slice boundaries. Each entry position is a range corresponding to the subslice being entered. The positions are chosen to minimize the net's global bounding box. Routes are determined using a pattern router.

Dai and Kuh [12] described a hierarchical method for combining floor planning and global routing for building-block layout. A bottom-up clustering followed by top-down floor plan enumeration is used to determine block positions. The global routing is performed during the top-down portion of the floor planning. At each level of the hierarchy, a particular floor plan template is chosen, and global routing is performed on connected subgraphs of the inner dual graph corresponding to the template's floor plan graph. Each net is routed by finding a minimum Steiner tree on this global routing graph. The edges and vertices of these trees are expanded to obtain the global routing graphs and the input and output pad goals for the next level.

The Mariner sea-of-gates system [11] interleaves placement steps and routing steps to improve layout quality. Though the placement algorithm is based on quadrisection [40] and the routing algorithm is based on 2×2 routing [10], both use more complicated data models that account for possible external pin positions. The interleaved global routing steps are used primarily to derive congestion metrics to direct the partitioning process. Final detailed placement is determined using simulated annealing and a maze router is used to obtain final global routes.

Korte, Prömel, and Steger [3] used a graph partitioning paradigm to combine partitioning (global placement) and global routing for sea-of-gates designs. The algorithm combines the tasks of partitioning cells into clusters, assigning these clusters to regions on the chip image, and providing the global routes with respect to these regions. The partitioning is a recursive min-cut bi-partitioning procedure and the global routing is solved as the problem of packing Steiner trees for various terminal sets in a graph with specified edge capacities. The global routing algorithm handles partially routed nets so at each iteration global routes are only updated in correspondence to the local change in the global routing graph. The partitioning cuts are chosen to be the "most critical" for the global routing and the partitioning cost function encodes the global routing information.

Suaris and Kedem [41] built a standard-cell layout system that combines their quadrisection placement algorithm with loose global routing in order to minimize the total layout area. The global routing is divided into two phases. The *loose global routing* phase

is combined with the placement and determines spanning trees for each net. The *detailed global routing* phase is performed separately and finds the actual global routes for each net. Finding these actual global routes includes determining the feed-through locations for nets to cross intermediate rows. Placement and loose global routing are performed as sequential operations for each problem abstraction of the quadrisection hierarchy and terminal propagation is used to feed forward loose global routing information to guide placement on subsequent abstractions. They are working on making the detailed global routing hierarchical so that it may be combined with the placement. The new detailed global router will find Steiner trees rather than spanning trees for each global route.

Müller [27] proposed a hierarchical method for combining floor planning and global routing. A min-cut partitioning process will be used to build a cut tree of the circuit. The partitioning will stop when the number of modules in a partition is smaller than a certain threshold. The partitioning structure of the remaining subcircuits is to be determined bottom-up with a clustering technique based on a maximum weighted matching algorithm. Then, the cut tree will be processed bottom-up to determine cut line orientation and to estimate required routing area. The routing area estimation is expected to be a combination of statistical methods and actual global routing computations. A final top-down pass over the cut tree will be performed to determine partition ordering with respect to each cut line, to distribute routing area, and to actually construct the global wiring.

Ohmura, et al. [29] presented a hierarchical floor planning method that also determines global routes. Modules or cells are defined as either *hard* or *soft*. Hard modules have a defined shape and size (area). Soft modules have a defined size, but an unknown shape. The approach consists of two phases. First, an initial floor planning phase is performed to determine the basic topology of hard modules and sets of soft modules. It also allocates routing area for critical wires. Second, a detailed floor planning phase is executed to partition the sets of soft modules, to determine positions for the soft modules, and to assign global routes. The partitioning is performed so that the congestion of switchboxes and channels on the next level of hierarchy is reduced. The heuristic used is to partition each rectangular region such that the ratio of subpartition areas is equal to the ratio of nets entering the subpartitions. Only nets that enter a subpartition from outside the rectangular region and cross over an edge of the rectangular region that is parallel to the partition are considered for this metric.

Recently, Bapat [4] presented a method of merging placement and routing based on

a 3×3 grid data model. The method computes and stores Steiner trees on the 3×3 grid for later table look-up. The solution of the 3×3 placement and routing abstraction starts with a random initial partition followed by random assignment of minimal length Steiner trees. Next, an algorithm similar to the Kernighan-Lin-Fiduccia-Mattheyses [19, 15] partitioning method is used to reduce net length through cell movements, to reduce congestion imbalance through selection of alternative minimum length Steiner trees, and to reduce congestion imbalance through cell movements. Problem abstractions within a level of hierarchy are solved independently and then the 3×3 algorithm is applied to abstractions formulated across the original decomposition boundaries to combine the partial solutions.

Shragowitz, Lee, and Sahni [36] developed a system to combine placement and global routing for sea-of-gates design styles. A constructive placement is performed one slice at a time from left to right. Slices are connected subareas of the set of unused cell positions and need not be uniform or symmetric in either the horizontal or vertical directions. At each step, nets that connect the new slice and the previously placed portion of the circuit are assigned global routes. A maze router is used for the global routing and the position and size of the new slice may be adjusted to accommodate the routing.

Tzeng [43] implemented a method of combining routing and placement adjustment. Wiring results are stored in a global data structure and cell positions are maintained using a global triangulation graph. An initial constructive placement is derived using standard methods. Next, routing and placement adjustment are iterated. The router is a sophisticated area router based on a hierarchical $2 \times n$ routing paradigm. The router is able to re-route existing wiring assignments, so its input may contain overlapping wires and partial routes. Placement adjustment is performed by using a one dimensional compaction algorithm on the triangulation graph. Congestion metrics from the routing are translated into spacing constraints on the graph. After placement adjustment, special care is taken to preserve as much of the previous wiring as possible. The resulting routing problem with possible conflicts and broken connections is presented to the router for another iteration. A final compaction step is performed using a more sophisticated compaction algorithm than in the placement adjustment phase.

2.3 Unified Placement and Routing

The unified placement and routing problem remains intractable because of the size and complexity of current circuit designs. However, Burstein reported some attempts at solving the unified problem. His work consists of a basic technique for performing placement and routing in a plane [7] and a partitioning method to handle multiple wiring layers [8].

The basic technique consists of forming a planarized image of the circuit graph, converting this image into an annular embedding, and then finding a geometric realization of the circuit corresponding to the embedding. The last step is formulated as the problem of finding a system of polyominoes that corresponds to the annular embedding such that the bounding box of the system is minimized. Channel or river routing must still be performed between the layout rings. The planarization step uses a heuristic $O(m^2)$ algorithm (where m is the number of edges in the circuit graph) that attempts to minimize the crossing number of the graph. If input and output pins are required to be along chip boundaries, certain embedding choices may be influenced by the distribution of these pins. This method is interesting, but impractical for current circuits. Neither the planarization step nor the reduction to an annular embedding directly address layout constraints or goals and the transformations from planarized image to geometric implementation are overly restrictive.

The basic technique is extended to multiple layers by partitioning the circuit into as many planar parts as there are wiring layers. When possible, nets are routed using a single wiring layer; otherwise, vias are introduced and are treated as circuit cells to determine their placement. Each plane is processed using the basic technique. Consistent placement of cells and vias between planes can be guaranteed only if two wiring layers are used and if the embedding on one of the layers is acyclic.

2.4 Summary

Merged approaches are interesting compromises in the attempt to integrate placement and routing phases. A unified approach would be ideal, but current unified problem formulations are either too simplistic or too complex to be practical. Coupled approaches have utility, but must restrict the possible placement modifications to avoid excessive re-routing. These restrictions place a premium on the performance of the initial placement phase which must execute in the absence of routing information. The placement adjust-

ments primarily re-space cells with out changing the basic cell topology much. Merged approaches are interesting because they can cover a wide range of integration possibilities. This characteristics provides a flexible structure for investigating the relationship between placement and routing and for exploring the possibility of solving a truly unified placement and routing problem. At one extreme, a merged approach can mimic a coupled approach by restricting permissible placement modifications. At the other extreme, a merged approach can solve placement and routing concurrently with the routing solution starting simultaneously with the placement solution. In between, lie many possibilities for placement and routing interaction and communication. Ideally, this interaction and communication should be used to avoid poor layout decisions or at least, to organize and structure the layout process for effective identification of poor decisions. Unfortunately, existing merged methods do not adequately emphasize this communication between phases. They may use routing information to influence subsequent placement decisions, but never use it to change previous decisions. Communication is further inhibited by the differences in data models used by each phase, and routing results are not always easily translated into new placement decisions.

The layout system presented in the following chapters is a merged hierarchical approach based on a common 2×2 grid-graph data model. The hierarchical framework is used to analyze the relationship between placement and routing more closely than previous work and to investigate and compare different integration possibilities. The use of a common data model is intended to alleviate the communication problems of previous approaches and to allow the implementation of effective feedback mechanisms.

Chapter 3

Algorithms

The basic algorithmic approach of this research to solving the layout problem is to reduce or divide the original problem into smaller or simpler subproblems, solve the subproblems, and then combine the partial solutions to form an entire solution to the original problem. This approach is often referred to as a *problem reduction* or *divide-and-conquer* method [2] and techniques of this class are often used to deal effectively with large and complex problems. Within this framework, specific algorithms are used to solve the layout subproblems and the constraints between subproblems and to implement a rip-up and re-route capability.

Hierarchical decomposition is the particular form of divide-and-conquer technique implemented. Its distinguishing characteristics are that the original problem is decomposed into a set of subproblems of the same form and that the decomposition process is performed recursively on each subproblem until the subproblems become simple enough to solve immediately. A novel feature is that the hierarchical decomposition for all aspects of the layout problem uses the same 2×2 grid-graph data model.

Decomposition of a subproblem into the next set of subproblems requires solving the layout problem for the corresponding 2×2 abstraction. A quadrissection [40] style algorithm is used to solve the 2×2 placement or partitioning problem and a linearly constrained nonlinear integer program is solved to generate the 2×2 routing. This route assignment algorithm is an extension of Burstein's switchbox router [10].

Constraints between subproblems must be enforced in order to guarantee that partial solutions may be combined into a physically possible solution. These constraints are introduced through the use of special pin-pairs along the subproblem boundaries. These ter-

minals are called *pseudo-pins* and the specification of their locations represents the required constraints on the partial solutions. The problem of determining pseudo-pin locations is called the *pseudo-pin assignment* problem.

The hierarchical decomposition is a constructive approach and the rip-up and re-route capability implemented is one example of a structured method for finding and modifying poor heuristic decisions. The procedure is based on the same 2×2 abstraction as the other algorithms.

3.1 Hierarchical Decomposition

The hierarchical decomposition performs problem reductions of the layout problem based on a 2×2 data model. The reduction continues recursively until the grids of the current data model abstraction correspond exactly to the desired final layout grid. Each reduction is based on the solution of the current abstraction.

3.1.1 Common Data Model

The data model used for the hierarchical decomposition of the layout area is the 2×2 grid-graph. This graph was chosen as the simplest useful abstraction for layout sub-problems. Using the same model for all aspects of the layout process simplifies management of the interaction between phases and increases the opportunities for combining phases.

The 2×2 data model is depicted in Figure 3.1 as both a set of grid cells and the standard grid-graph representation. The 2×2 grid-graph is the graph $G(\mathcal{V}, \mathcal{E})$, consisting of vertices

$$\mathcal{V} = \{v_1, v_2, v_3, v_4\}$$

and undirected edges

$$\mathcal{E} = \{e_1, e_2, e_3, e_4\}$$

such that:

$$e_1 = (v_1, v_4),$$

$$e_2 = (v_2, v_1),$$

$$e_3 = (v_2, v_3),$$

$$e_4 = (v_3, v_4).$$

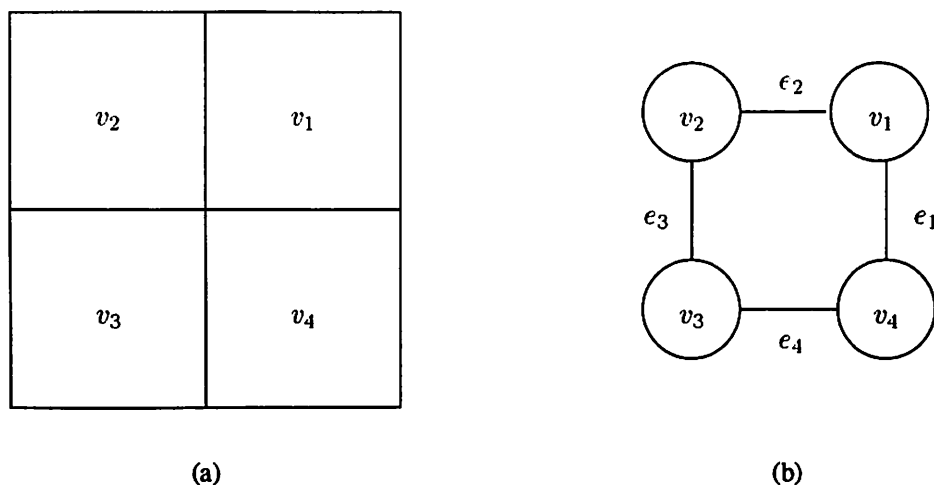


Figure 3.1: The 2×2 data model represented as (a) a 2×2 grid of cells and (b) the corresponding grid-graph.

This labeling of vertices and edges will be used in all subsequent discussion.

Choosing a data model involves compromising between simple problem formulation and useful progress towards a final solution. The 2×2 grid-graph is the simplest model that captures enough information to make two dimensional layout decisions. The model can be used to perform all aspects of layout from placement through routing, including feedback between phases, rip-up and re-place, rip-up and re-route, and pseudo-pin assignment on subproblem boundaries.

Improved layout quality requires effective communication between layout phases. Better communication is possible with a common data model because it provides a common reference frame for the different layout phases. A common reference frame simplifies and enhances information flow by reducing the amount of translation and interpretation required between phases. Using the same data model for all the layout phases provides opportunities for integrating the different layout phases in that information from each phase can be obtained for each subproblem abstraction concurrently.

3.1.2 Problem Reduction

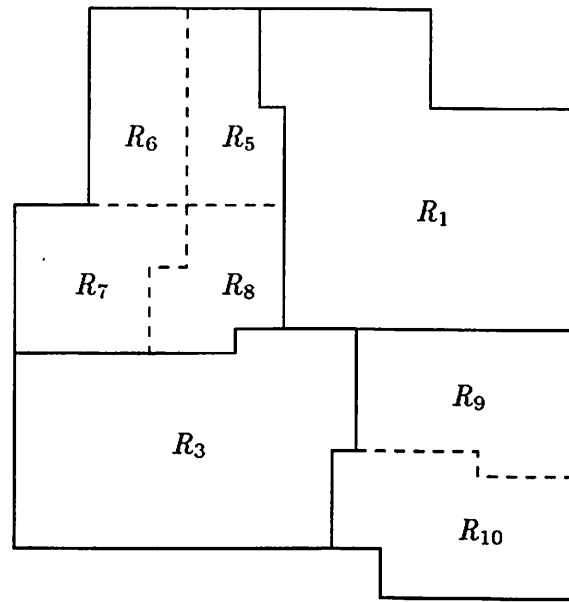
Problem reduction techniques create a tree or hierarchy of subproblems. Characterizing the relationship between the parts of this problem hierarchy and the original problem is useful for analyzing and describing algorithm features and effects. In these

experiments, the actual reduction is based on formation and solution of 2×2 layout abstractions and corresponds to a decomposition of the layout region into subregions. An interesting feature of the problem hierarchy is that each subsequent level of subproblems represents a more detailed view of the original problem.

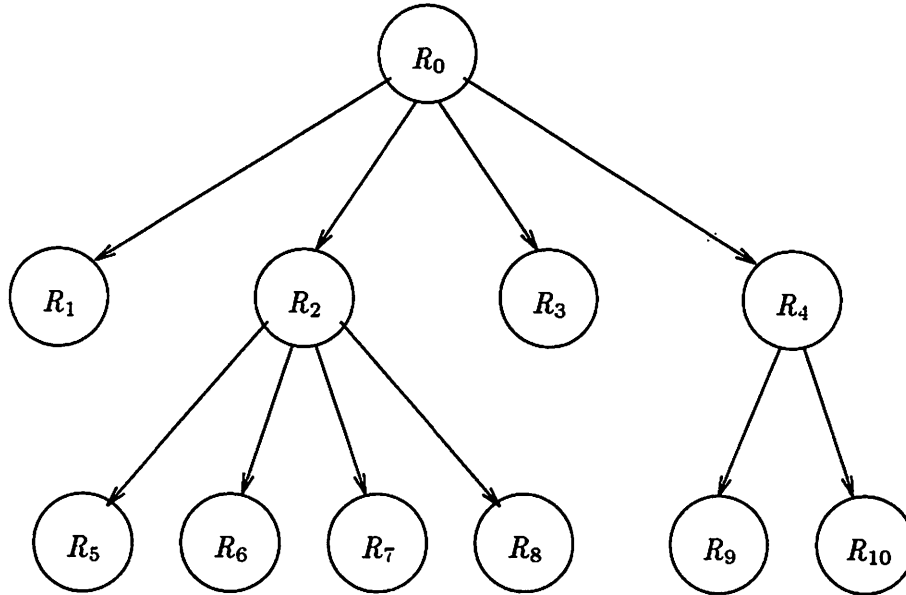
Hierarchy Hierarchical decomposition creates a problem hierarchy where each node represents the data model abstraction of the portion of the original problem corresponding to the subtree rooted at that node. This relationship is illustrated in Figure 3.2. The basic decomposition step physically partitions the region allocated to the original circuit into smaller subregions. The original problem corresponds to the root node, R_0 , of the problem hierarchy. At the first decomposition step or level, the region of the original problem is divided into four subregions, corresponding to the four children subproblems R_1 , R_2 , R_3 , and R_4 . At the next level, subproblem R_2 also is divided into four subproblems, R_5 through R_8 , while subproblem R_4 is reduced to two subproblems, R_9 and R_{10} .

Decomposition The decomposition or reduction of a layout problem into simpler subproblems requires forming an abstraction of the layout problem based on the data model, solving the abstraction, and then using the solution to create the appropriate subproblems. In this research, 2×2 abstractions are formed based on a partitioning of the layout region into subregions. The solution of the abstraction depends on the particular application of the hierarchical decomposition method. Separate placement and routing phases can be implemented using this method or because the same data model is used for both phases, a combined placement and routing phase can be implemented. In the case of separate phases, subproblems are generated by solving the relevant aspect of the layout problem. However, in the following example, solutions to both placement and routing are used to illustrate the decomposition of the layout problem in general. Furthermore, in the case of routing, more information is desired for the generation of subproblems than is specified by the 2×2 routing solution. Extra constraints on net crossing positions must be added to allow feasible combination of partial routing solutions.

The 2×2 grid abstraction of a layout region depends on the determination of subregion shapes and sizes. In general, hierarchical decomposition does not require grid assumptions or straight partition boundaries [24], but in these experiments, for simplicity and efficiency, an underlying grid is assumed and straight line segments are used to divide



(a)



(b)

Figure 3.2: Hierarchical Decomposition. (a) The decomposition of a layout region and (b) the corresponding problem hierarchy.

regions between grid positions. In general, two line segments are chosen to divide a region into four subregions, corresponding to the four grid cells of the 2×2 abstraction. One line segment partitions the region vertically and the other segment partitions the region horizontally. The line segment partitions are chosen to generate subregions of approximately equal area. This leads to a problem hierarchy of shorter height and a smaller total number of subproblems, reducing the amount of required computation. If the horizontal span of a region ranges from grid 0 to grid $w - 1$, then a vertical line segment that lies between grid $i - 1$ and grid i is chosen, where

$$i = \left\lceil \frac{w}{2} \right\rceil.$$

An analogous calculation is performed to find the position of a horizontal line segment that partitions the region vertically. A value of zero for i indicates a degenerate abstraction. Degenerate 2×2 abstractions are problems that are really 1×2 or 2×1 grids. A degenerate abstraction may be created intentionally depending on the aspect ratio of the problem region. If the height of the region is greater than or equal to twice the width of the region, then only a horizontal line segment will be chosen to partition the region vertically. Similarly, if the width of the region is greater than or equal to twice the height of the region, then only a vertical line segment will be chosen to partition the region horizontally.

Figure 3.3 illustrates the decomposition of a layout problem into its constituent subproblems. Each decomposition step must reduce a problem to a set of subproblems of the same form. In general, this involves partitioning the layout region into subregions, assigning cells to each subregion, determining subnets for each subregion, and adding boundary consistency constraints so that subproblem solutions can be combined to form the whole solution.

Figure 3.3 (a) shows a small example problem. The problem has five cells and nine nets. The cells are drawn with arbitrary positions for clarity and the curved lines connecting cells represent the unrouted nets. Cells A and B are labeled and the net connecting them is drawn with a different line style to help illustrate the decomposition process.

Figure 3.3 (b) shows the result of performing placement on the problem abstraction. Each cell has been assigned to one of the four grid cells as drawn. For example, cell A is assigned to the lower-left grid cell and cell B is assigned to the upper-left grid cell.

Figure 3.3 (c) shows the result of performing routing on the problem abstraction. The routing problem abstraction is formed by mapping each net of the original problem

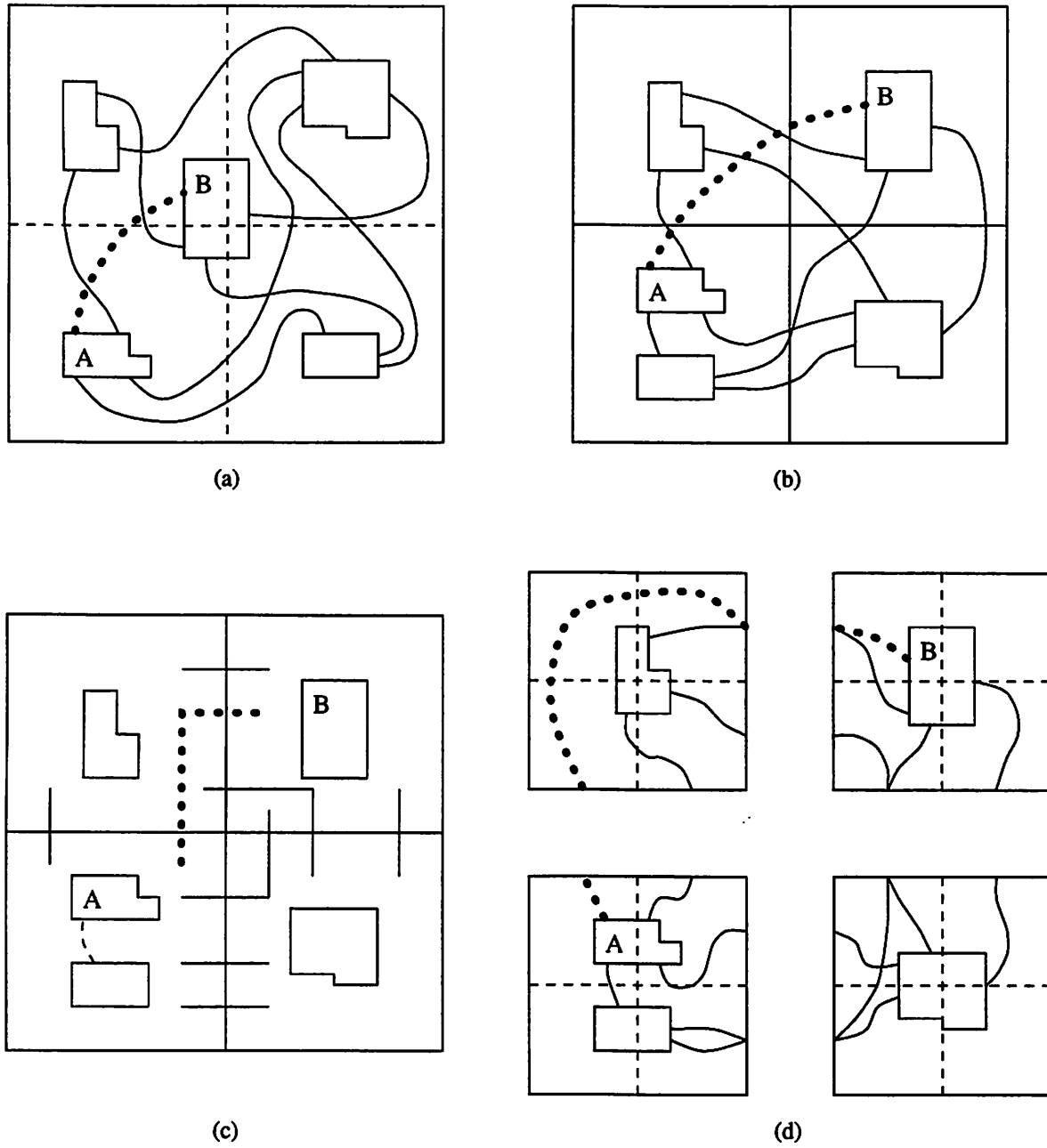


Figure 3.3: Example decomposition step

in Figure 3.3 (a) into a corresponding abstract net. Since there are four possible locations on the 2×2 grid, abstract nets may have one, two, three, or four terminals. An abstract net has a terminal located in a grid cell if the original net has at least one terminal in the corresponding subregion. Abstract nets with only one terminal are considered degenerate with respect to the current abstraction and are ignored. The net connecting cell A to its companion cell in the lower-left grid cell is an example of a degenerate net. In the figure, route assignments are represented by line segments that are drawn crossing the grid cell boundaries.

Figure 3.3 (d) shows the final reduction into subproblems. Each subproblem is a layout problem for the subregion represented by its corresponding grid cell. The subcircuit description for each subproblem is derived from the placement and routing results of the current abstraction. The cells of a subproblem are those cells assigned to its corresponding grid cell. Nets for each subproblem are derived from the route problem abstraction and the route assignments. Also, extra constraints are added on some of the pins of the subproblem nets so that the final solution may be realized. The reduction results in subproblems that have the same form as the original problem.

Subnets are generated from both the degenerate and nondegenerate nets of a problem. Degenerate nets of the current abstraction are nets of the subproblem corresponding to their terminal's location. For example, the degenerate net connected to cell A is a net of the lower-left subproblem. Whether or not degenerate nets are also degenerate with respect to their new subproblem abstraction depends on the subsequent placement. Other subproblem nets are derived from the current abstraction's route assignments. If the route of a net uses a grid cell, then a subnet corresponding to that portion of the route is present in the corresponding subproblem. For example, in Figure 3.3 (c) the net connecting cell A and cell B is assigned a route that uses the upper-right, upper-left, and lower-left grid cells. In Figure 3.3 (d), this net has been divided into three subnets and each subnet is included in the appropriate subproblem. The pins of subnets are derived from two sources. First, some pins are from the original net specification and are contained in the subproblem as a result of region decomposition or cell placement. Second, some pins are added to represent connection to other subnets. These pins are referred to as *pseudo-pins* and mark where subnets enter and exit subproblems.

Consistency constraints must be enforced on subnet entry and exit locations in adjacent subproblems to ensure the feasibility of combining partial solutions to form a

complete solution. Consider the net connecting cell A and cell B in Figure 3.3 (d). In the final solution, the exit location of the net from the lower-left subproblem must be consistent with its entry location into the upper-left subproblem. Similar constraints exist for all net crossings of common subproblem boundaries. These consistency constraints are introduced by assigning pseudo-pins in pairs corresponding to net crossings. Both pins in a pair must always have the same position with respect to the axis of their assigned boundary. Determination of this position is discussed in Section 3.3. Based on the 2×2 routing solution, a pseudo-pin's final position may lie anywhere along the extent of its assigned boundary. In the figure, pseudo-pin locations have been refined just enough to determine grid positions on the next level of subproblem abstractions. Pseudo-pins assigned to a particular grid are drawn connected to the midpoint of the appropriate boundary.

Interpretation Hierarchical decomposition can be viewed as a gradual refinement of the layout solution. At higher levels of abstraction, solving the 2×2 placement problem can be thought of as solving the partitioning or global placement problem. Similarly, solving 2×2 routing problem abstractions solves global routing problems. Subsequent decomposition and abstraction in the hierarchy create more detailed views of the problem. Thus, the solution to each subsequent level of subproblems represents a further refinement of the solution to the original problem.

3.2 Subproblem Solution

The problem reduction step of the hierarchical decomposition depends on solving the layout problem for each 2×2 abstraction. Solving placement and routing simultaneously on each 2×2 abstraction would effect a unified approach to the layout problem. This possibility is discussed further in Section 6.1, but unfortunately, still remains intractable. Alternately, 2×2 placement and 2×2 routing may be solved separately. The level of placement and routing integration can be managed by using the problem hierarchy to control setup and feedback information for each subproblem abstraction. Solving placement and routing immediately for each subproblem implements a kind of merged placement and routing approach that solves the two phases concurrently, while solving the entire hierarchy of placement subproblems before solving the routing subproblems is equivalent to using separate phases.

3.2.1 Cell Assignment

Placement of a given circuit net list on the 2×2 grid-graph, $G(\mathcal{V}, \mathcal{E})$, requires that each cell be assigned to one of the four grid-graph vertices subject to admissibility constraints. Given a circuit description, let

$$\mathcal{M} = \{m_1, \dots, m_M\}$$

be the set of modules (or cells) and let

$$\mathcal{N} = \{n_1, \dots, n_N\},$$

be the set of nets. Placement is equivalent to partitioning \mathcal{M} into four partition blocks such that each block corresponds to a vertex of G . In this research, the partition is normally generated using a quadrisection algorithm similar to that described by Suaris and Kedem [40]. The exceptions occur for small, detailed abstractions where an exhaustive search is used instead.

Cell-to-vertex assignments are admissible if the set of cells assigned to a particular vertex does not exceed the vertex's limit. Each vertex corresponds to a particular region of the layout and the number of cells that may be placed legally within the region depends on the size and shape of the region and the size and shape of the cells. Cell size and shape are modeled as a single weight value. For a cell m_i , this value is denoted $weight(m_i)$. The area and design rule dependent constraints on legal cell placement are translated into limits on the total cell weight that can be assigned to each vertex. For a vertex v_j , this constraint is denoted $limit(v_j)$.

Partitioning

The partitioning algorithm assigns cells to vertices of G by dividing \mathcal{M} into four disjoint sets of cells such that each set corresponds to a vertex of G . If the set of cells assigned to v_j is denoted as \mathcal{M}_{v_j} , then a partition of cells can be represented as the vector

$$\vec{P} = (\mathcal{M}_{v_1}, \mathcal{M}_{v_2}, \mathcal{M}_{v_3}, \mathcal{M}_{v_4}).$$

Define the area of a partition block \mathcal{M}_{v_j} as

$$area(\mathcal{M}_{v_j}) = \sum_{m_k \in \mathcal{M}_{v_j}} weight(m_k),$$

and let the area limit for a partition block \mathcal{M}_{v_j} be the cell weight limit of v_j ,

$$\mathit{limit}(\mathcal{M}_{v_j}) = \mathit{limit}(v_j).$$

Also, associate with each cell is a set of legal partition block assignments,

$$\mathcal{L}_{m_i} = \{\mathcal{M}_{v_j} \mid \text{cell } m_i \text{ may be assigned to } v_j\}.$$

In these experiments, the weight value of a cell is the number of placement grid locations the cell occupies and the weight limit of a vertex is the total number of placement grid locations contained in its corresponding subregion.

For uniform cell sizes, the definitions given above are sufficient to satisfy the placement admissibility constraints, but extra constraints on the the distribution of area between partition blocks are often useful. Let

$$\mathit{target}(\mathcal{M}_{v_j})$$

be the desired area of \mathcal{M}_{v_j} , and let ϵ be the maximum error limit between the actual area of a partition block and its target value.

The objective is to minimize the partition cut weight

$$W(\vec{P}) = \sum_{n_i \in \mathcal{N}} w_{n_i}(t^{n_i})$$

where w_{n_i} is a weight function for the net n_i that depends only on t^{n_i} , the abstract net type of n_i . With respect to $G(\mathcal{V}, \mathcal{E})$, an abstract net type is a subset of \mathcal{V} and the abstract net type for a net n_i contains vertex v_j if the subregion corresponding to v_j contains at least one terminal of n_i .

Given a net list, the partitioning algorithm returns an optimal partition \vec{P}^* of minimal cut weight $W(\vec{P}^*)$ that satisfies:

$$\begin{aligned} \mathit{area}(\mathcal{M}_{v_j}) &\leq \mathit{limit}(\mathcal{M}_{v_j}) \quad \forall v_j, \\ \left| \mathit{area}(\mathcal{M}_{v_j}) - \mathit{target}(\mathcal{M}_{v_j}) \right| &\leq \epsilon \quad \forall v_j, \\ \mathcal{M}_{v_j} &\in \mathcal{L}_{m_i} \quad \forall m_i \in \mathcal{M}_{v_j}. \end{aligned}$$

The quadrisection partitioning algorithm is a min-cut [19] class algorithm. The algorithm starts by generating an initial partition of cells. Then, iterative improvement passes are applied to the partition until no further reduction in partition cut weight results.

```

initialize;
while ( a free cell can be moved ) {
    remove cell with largest gain from gain tables;
    lock cell in new partition block;
    update gains of affected cells;
    keep track of best partition;
}
use best partition found;

```

Figure 3.4: Basic quadrisection algorithm pass

Each improvement pass can be performed in $O(P)$ time where P is the number of pins in the circuit. In practice, the overall algorithm is quite fast because the number of required improvement passes is usually less than four or five.

As in all min-cut style algorithms, the quality of the final partition is strongly dependent on the starting partition. This algorithm generates an initial partition by performing min-cut bisection partitioning steps on the net list in both directions and then merging and balancing the results. The initial partition for each bisection is chosen using the following linear ordering method. First, all fixed cells are assigned to their appropriate partition blocks. Then, unassigned cells are assigned successively to a target block. The block with the largest number of fixed cells is chosen as the target block. If both partition blocks have the same number of fixed cells, then the target block is chosen randomly. At each step, the cell that shares the most nets with target block cells is selected. Cell selection stops when acceptable partition block area ratios are achieved. If the target partition block is initially empty, a seed cell is chosen randomly.

A pseudo-code description of the basic quadrisection algorithm pass is given in Figure 3.4. The quadrisection algorithm pass iteratively modifies a given partition by successively moving cells from one partition block to another. The number of iterations is controlled by allowing cells to move only once during each pass. Cells that have not moved are *free* and cells that have moved are *locked*. The *gain* of a cell move is defined as the decrease in partition cut weight resulting from that move. Rapid cell move selection is aided by the use of special gain tables to maintain an ordering of cell moves. At each

iteration, the move with the best gain is chosen to transfer a free cell to some partition block. The chosen move must be legal and the result of the move must satisfy the partition block area constraints described earlier. Cells are moved even if the move will result in a higher partition cut weight because the set of possible cell moves at each iteration is dependent on the previous cell moves taken. Choosing a cell move that increases partition cut weight may admit subsequent cell moves that will decrease partition cut weight by more than the increase. Iteration control can be further optimized while preserving this effect. Cell moves can be discontinued if the current lower bound on partition cut weight (or “locked” weight [20]) is greater than the best partition cut weight seen so far. The lower bound is calculated as that component of $W(\vec{P})$ contributed by *locked* nets. A net n_i is locked if no free cell moves exist that will change the abstract net type of n_i .

To maintain the linear time complexity of each algorithm pass, each net weight function w_{n_i} must depend only on the abstract net type of n_i and must evaluate in constant time. Also, the maximum net weight must be bounded by some constant z . These requirements are met by using weight function terms that are invariant during each execution of the quadrisection algorithm. The terms may be recalculated between quadrisection algorithm invocations to provide feedback to the placement process.

The net weight function is defined as

$$w_{n_i} = C_1 \sum_{j=1}^4 \text{weight}(v_j) \cdot \text{usage}_{t^{n_i}}(v_j) + C_2 \sum_{k=1}^4 \text{weight}(e_k) \cdot \text{usage}_{t^{n_i}}(e_k)$$

where $\text{weight}(v_j)$ and $\text{weight}(e_k)$ are vertex and edge costs, $\text{usage}_{t^{n_i}}(v_j)$ is an indicator variable for degenerate net types, $\text{usage}_{t^{n_i}}(e_k)$ represents the probability of edge e_k being used to route t^{n_i} , and C_1 and C_2 provide a relative weighting between vertex and edge costs. The usage value for a vertex with respect to an abstract net type is unity when the abstract net type is the degenerate net type corresponding to that vertex and zero otherwise. The usage value for an edge with respect to an abstract net type, $\text{usage}_{t^{n_i}}(e_k)$, is the average number of times an edge is used per route in the optimal set of routes for the abstract net type. For these purposes, an optimal route is a minimal length route, so

$$\mathcal{R}_{t^{n_i}}^* = \{r \mid r \text{ is a minimal length route for } t^{n_i}\}$$

and

$$\text{usage}_{t^{n_i}}(e_k) = \frac{\sum_{e_k \in r, r \in \mathcal{R}_{t^{n_i}}^*} 1}{|\mathcal{R}_{t^{n_i}}^*|}.$$
















Type, t	$usage_t(e_3)$	$usage_t(e_2)$	$usage_t(e_1)$	$usage_t(e_0)$
	0	0	0	0
	0	0	0	0
	0	0	1	0
	0	0	0	0
	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
	0	1	0	0
	0	1	1	0
	0	0	0	0
	0	0	0	1
	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
	0	0	1	1
	1	0	0	0
	1	0	0	1
	1	1	0	0
	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$

Table 3.1: Relative edge weights by abstract net type and edge

Values for $usage_{t^i}(e_k)$ are shown in Table 3.1. The values of other cost function components are discussed later.

Exhaustive Search

An exhaustive search method is used on small partitioning problems. The method uses exhaustive enumeration in combination with heuristic pruning strategies and is invoked only on problems with solution space sizes less than or equal to 128. This limit was chosen empirically to optimize the trade-off between solution quality and run time. The pruning techniques use knowledge of the enumeration order to prune sequences based on cost lower

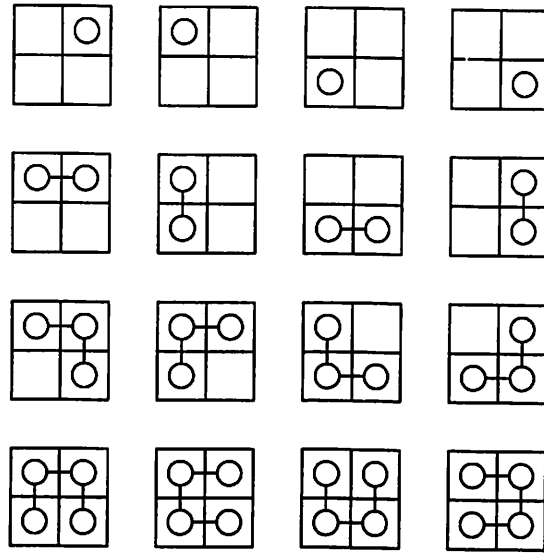


Figure 3.5: Possible routes on the 2×2 abstraction

bounds and feasibility conditions. The pruning strategies are heuristic in that problems exist that will not have any states pruned. However, execution time benefits from any pruning and the average percentage of states pruned during the solution of a given circuit example range from 53% to 64%.

3.2.2 Route Assignment

Routing a given circuit net list on the 2×2 grid-graph, $G(\mathcal{V}, \mathcal{E})$, requires assigning each net a route pattern or route configuration such that all assignments are admissible. The assignment problem is solved in two phases. First, nets are classified into types based on the location of their pins with respect to the cells of the 2×2 grid abstraction and route patterns are assigned to sets of nets of the same type. Second, the route patterns assigned to each set are assigned to specific nets within that set.

Route-to-net assignments are admissible if all nets are routed and all routing resource constraints are satisfied. Nets are classified into abstract net types with respect to $G(\mathcal{V}, \mathcal{E})$ as in Section 3.2.1. A route configuration $G'(\mathcal{V}', \mathcal{E}')$ is a subtree of G and connects or routes a net n_i if \mathcal{V}' is a superset of the abstract net type of n_i . Figure 3.5 shows an enumeration of all possible routes for G . Routing constraints are modeled as edge and vertex

capacities in G . Edge capacities represent the number of usable routing tracks and columns and vertex capacities represent the number of usable via locations. Assigning route G' to a net is equivalent to embedding G' in G . The wiring for a net that is assigned a route G' is assumed to change both direction and layer in the subregions corresponding to vertices of G' that have two adjacent edges. When G' is embedded in G , each edge of G' uses a unit of capacity of its corresponding edge in G , and each vertex of degree two in G' uses a unit of capacity of its corresponding vertex in G . A set of embeddings is admissible if edge and vertex usage does not exceed capacity limits. Given capacity values for all edges and vertices in G , a set of route-to-net assignments is admissible if the corresponding set of embeddings is admissible.

Net Type Assignment

The problem of assigning routes to groups of nets is formulated as a linearly constrained nonlinear integer program. The major differences from the formulation of Burstein and Pelavin [10] are the use of a nonlinear cost function term to distribute route patterns more evenly and the assignment of routes to all nets despite resource constraint violations. The nonlinear formulation generates a better distribution of wiring patterns and is less than twice as slow as the corresponding linear formulation. The complete assignment of routes is used to determine congested areas if a subsequent re-route phase is required.

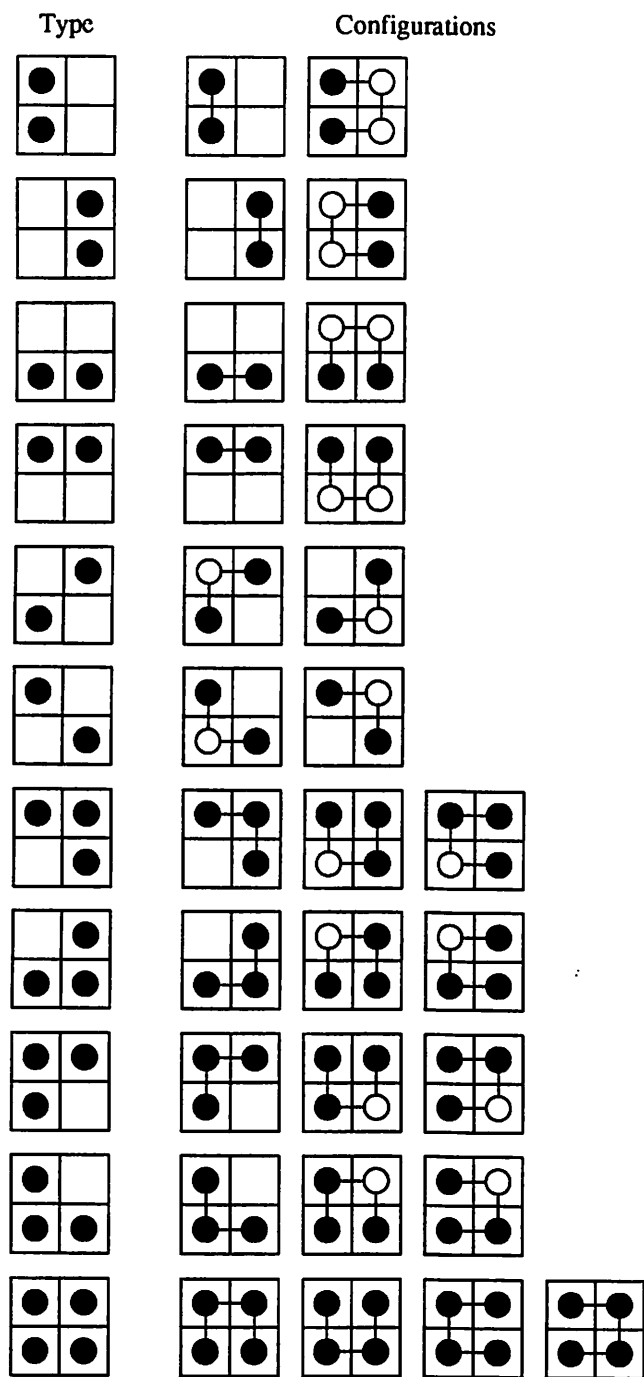
Constraints The constraints on the integer program must reflect the assignment constraints and the capacity constraints of the 2×2 route assignment problem.

Given the set of possible route configurations $\mathcal{R} = \{r_j\}$ and the set of possible net types $\mathcal{T} = \{t_1, \dots, t_T\}$, let

$$R_{t_i} = \{r_j | r_j \text{ is a permissible route for type } t_i\}$$

be the permissible route patterns for net type t_i . For nondegenerate nets, the possible types and their corresponding sets of possible routes are shown in Figure 3.6. Membership in these sets can be restricted if desired. Also, let:

$$\begin{aligned} N_{t_i} &= \text{the number of nets of type } t_i, \\ x_{t_i, r_j} &= \text{number of nets of type } t_i \text{ assigned route } r_j, \\ \text{capacity}(e_k) &= \text{the capacity of edge } e_k, \end{aligned}$$

Figure 3.6: 2×2 net types and configurations

$via_capacity(v_k)$ = the via constraint on vertex v_k .

The assignment constraints can be stated as:

$$\begin{aligned} x_{t_i, r_j} &\geq 0, & t_i = 1, \dots, T, r_j \in R_{t_i}, \\ \sum_{r_j \in R_{t_i}} x_{t_i, r_j} &= N_{t_i}, & \forall t_i. \end{aligned}$$

The non-negativity constraints prohibit physically impossible solutions that have negative numbers of route patterns assigned to net types. The other constraints guarantee one-to-one assignment of nets to route patterns appropriate for each net's type and assign a full set of route patterns to each net type regardless of possible capacity violations. These assignments represent the desired routes for each net type and if capacity violations occur, are used to identify congested regions suitable for re-routing.

The capacity constraints are represented as:

$$\begin{aligned} s_{e_k}^+ - s_{e_k}^- + \sum_{i=1}^T \sum_{\{j|e_k \in r_j, r_j \in R_{t_i}\}} x_{t_i, r_j} &= capacity(e_k), & \forall e_k, \\ s_{v_k}^+ - s_{v_k}^- + \sum_{i=1}^T \sum_{\{j|v_k \in r_j, r_j \in R_{t_i}\}} x_{t_i, r_j} &= via_capacity(v_k), & \forall v_k, \\ s_{e_k}^+, s_{e_k}^- &\geq 0, & \forall e_k, \\ s_{v_k}^+, s_{v_k}^- &\geq 0, & \forall v_k. \end{aligned}$$

The first line of constraint equations reflect edge capacity constraints and the second line represents via or vertex constraints. Capacity values, $capacity(e_k)$ and $via_capacity(v_k)$, are calculated using heuristics from Burstein and Pelavin [10].

The quantity

$$s_{e_k}^+ - s_{e_k}^-$$

is the difference between the capacity of edge e_k and the utilization of edge e_k . Since this value is unrestricted, it is modeled as the difference of two restricted variables, $s_{e_k}^+$ and $s_{e_k}^-$ where $s_{e_k}^+$ represents the excess capacity of edge e_k and $s_{e_k}^-$ represents the over-utilization of edge e_k . Similarly, the difference between vertex capacity and usage for vertex v_k ,

$$s_{v_k}^+ - s_{v_k}^-,$$

is modeled as the difference of restricted variables $s_{v_k}^+$, the excess capacity at v_k , and $s_{v_k}^-$, the over-utilization of vias at v_k .

These constraints do not actually force the integer program to find admissible solutions. Instead the variables in these constraints are incorporated into the objective function so that an admissible solution will be found if possible. If the problem is infeasible, a complete route assignment is still produced to aid identification of congested areas for re-routing.

Objective Function The objective is to minimize

$$\begin{aligned} cost = & - \sum_{k=1}^4 (L_{e_k}^+ s_{e_k}^+ - L_{e_k}^- s_{e_k}^-) - \sum_{l=1}^4 (L_{v_l}^+ s_{v_l}^+ - L_{v_l}^- s_{v_l}^-) \\ & + D \sum_{i=1}^3 \sum_{j=i}^4 \left(\frac{usage(e_i)}{capacity(e_i)} - \frac{usage(e_j)}{capacity(e_j)} \right)^2 \end{aligned}$$

where

$$usage(e_k) = capacity(e_k) - (s_{e_k}^+ - s_{e_k}^-).$$

The coefficients $L_{e_k}^+$, $L_{e_k}^-$, $L_{v_l}^+$, $L_{v_l}^-$, and D are chosen to implement the following goals in order of priority.

1. Avoid capacity violations.
2. Minimize net length.
3. Distribute edge usage evenly.

The linear terms represent the capacity violation and net length cost. In general, the shortest route is always chosen unless the choice would violate an edge capacity constraint. If the shortest route will violate a capacity constraint, then a longer route is assigned unless the assignment would violate an edge constraint. Since via capacity limits are calculated conservatively, the coefficients have been set to allow assignment of a longer route even if it results in a via capacity violation. Currently, the linear term coefficients are set as follows:

$$\begin{aligned} L_{e_k}^+ &= 2 \quad \forall e_k, \\ L_{e_k}^- &= 13 \quad \forall e_k, \\ L_{v_k}^+ &= 2 \quad \forall v_k, \\ L_{v_k}^- &= 4 \quad \forall v_k. \end{aligned}$$

The nonlinear cost function term is used to avoid assignments similar to Figure 3.7 (a). Assuming all boundaries have equal capacity, the even distribution of routes

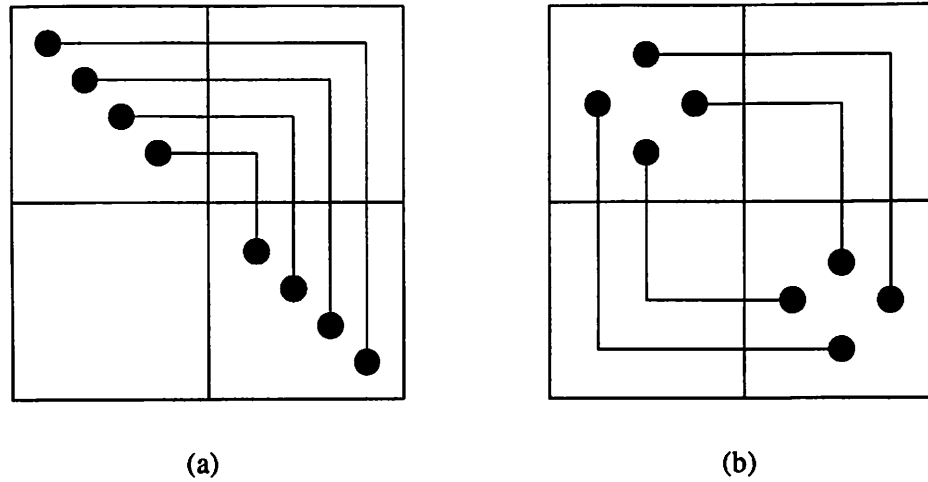


Figure 3.7: Assuming uniform and sufficient capacity, the uneven use of capacity (a) is undesirable, and the even distribution of routes (b) is preferred because of its more favorable effect on subproblem congestion.

reflected in Figure 3.7 (b) is preferred because it usually produces less routing congestion in subsequent subproblems. Even distributions can be favored by minimizing the square of the difference in usage-to-capacity ratios for all pairs of edges in the 2×2 grid-graph. In Figure 3.7, the benefit of even distribution is gained without any increase in net length. Since the suggested distribution metric does not account for net length, the square terms are weighted by an appropriate factor in the objective function so that the criterion is applied only when a more even distribution of routes will not increase total net length. This is implemented by setting the distribution weight factor, D , for each abstraction so that the distribution term is less than unity.

Program Characteristics Excluding non-negativity conditions, the problem has

$$\begin{aligned} T + E + V &= 11 + 4 + 4 \\ &= 19 \end{aligned}$$

constraints and with respect to the routes enumerated in Figure 3.6, it has

$$\begin{aligned} |\{x_{t_i, r_j}\}| + |\{s_{e_k}^+\}| + |\{s_{e_k}^-\}| + |\{s_{v_l}^+\}| + |\{s_{v_l}^-\}| &= 28 + 4 + 4 + 4 + 4 \\ &= 44 \end{aligned}$$

variables. As in the work of Burstein and Pelavin [10], the size of the programming problem is constant and independent of the number of nets in the routing problem, and classification of nets into types requires time linear in the number of nets. In practice, the solution of this linearly-constrained, nonlinear integer program has been quite efficient. Integer solutions are found using the branch-and-bound technique described by Papadimitriou [30]. At each decision point, a linearly constrained nonlinear program is solved using the MINOS optimization system [28]. A special C++ interface to MINOS has been written to setup and call the internal FORTRAN routines directly instead of writing input files and reading output files. As described in the next section, experimental results indicate that the solution to the initial linearly constrained nonlinear program is often integer or close to integer and that typically, few steps of the branch-and-bound algorithm are necessary to generate integer solutions. This is similar to the results found by Raghavan [31] for a 0-1 integer program formulation of the routing problem.

Results The performance of the 2×2 route assignment as a separate routing phase was compared with and without the nonlinear cost function term on the benchmark examples. Difficult routing problems were created from the circuit net lists by using the 2×2 cell assignment in a separate placement phase to position the sea-of-gates cells in a *minimum area* grid with aspect ratio as close to unity as possible. Details of the separate phase algorithms are described in Chapter 4. Table 3.2 compares results and program execution times for the router with and without the nonlinear cost function term.

Routing quality is measured by the number of edge capacity violations in the final global routing graph. Each unit of edge over-utilization counts as a violation. The nonlinear objective function worked well at improving routing quality. Using the nonlinear objective function always reduced the number of capacity violations. On some circuits the improvement is dramatic. For example, on circuit **C6288**, the number of violations dropped from 189 to 13.

Execution time is measured in CPU seconds on a DECSystem 5900/260. The results were obtained without the benefit of performing a re-routing phase in order to guarantee that comparisons between the linear and nonlinear objective functions for the same circuit example are based on the same number of integer programs. When using the nonlinear objective function, execution time increased by a factor which varied from 1.18 to 1.96.

Example	Net Count	Capacity Violations		Execution Time		
		Linear	Nonlinear	Linear,(s)	Nonlinear,(s)	$\frac{\text{Nonlinear}}{\text{Linear}}$
misex3c	309	4	0	46	90	1.96
C880	317	10	5	37	70	1.89
duke2	321	2	0	47	82	1.74
C1355	333	13	9	39	71	1.82
misex3	358	4	0	54	95	1.76
C1908	359	12	6	47	82	1.74
C3540	834	71	46	166	248	1.49
C5315	1251	1255	1106	418	495	1.18
C7552	1572	857	589	521	647	1.24
C6288	2417	189	13	770	1153	1.50

Table 3.2: Comparison of linear versus nonlinear objective function

Objective Function	Low	Mean	High
Linear	1	1.03	19
Nonlinear	1	2.74	99

Table 3.3: Statistics on the number of non-integer programs solved during the branch-and-bound solution of integer programs

Table 3.3 compares statistics for the number of linear programs or linearly constrained nonlinear programs solved during the branch-and-bound solution of integer programs. Over the course of solving all the examples, 7039 integer programs were solved for each objective function option. The integer linear program formulation has a structure such that solving the corresponding linear program almost always results in integer solutions. For these examples, the largest number of linear program solutions required is 19, but the average number is very close to one. Fortunately, much of this structure remains unaltered after the addition of the nonlinear objective function term. On many of the linearly constrained nonlinear integer programs, solving the corresponding linearly constrained nonlinear program also results in integer solutions. Though the largest number of linearly constrained nonlinear program solutions required is 99, the average number is only 2.74. This value is 2.66 times the average for the linear objective function formulation and is consistent with the execution time results in Table 3.2.

Individual Net Assignment

The solution to the integer program described above only assigns routes to net types and not to individual nets. To complete the route assignment, the assigned routes must be assigned to individual nets within each group. For each group, this problem is solved as a linear assignment problem with the goal of minimizing net length. These problems are defined with respect to the previous integer program as follows.

Let

$$\mathcal{N}' = \{n'_1, \dots, n'_{N_{t_i}}\}$$

be the set of nets of type t_i and define:

$$\begin{aligned} \text{cost}(n'_i, r_j) &= \text{the cost assigning route } r_j \text{ to net } n'_i, \\ y_{n'_i, r_j} &= \begin{cases} 1 & \text{if net } n'_i \text{ is assigned route } r_j \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Minimize

$$\sum_{n'_i \in \mathcal{N}', r_j \in R_{t_i}} \text{cost}(n'_i, r_j) \cdot y_{n'_i, r_j}$$

subject to:

$$\begin{aligned} \sum_{n'_i \in \mathcal{N}'} y_{n'_i, r_j} &= x_{n'_i, r_j} \quad \forall r_j \in R_{t_i}, \\ \sum_{r_j \in R_{t_i}} y_{n'_i, r_j} &= 1. \end{aligned}$$

This assignment problem can be solved as a minimum cost flow problem [22] with an algorithm implementation that runs in $O(N_{t_i}^4)$ time. The cost of assigning route r_j to net n'_i is calculated as an estimate of the wire length of net n'_i resulting from using route r_j . Route configurations that detour across an axis have an extra length term which penalizes these patterns if the terminals of the net are farther away from the axis. All lengths are estimated with respect to the final detailed routing grid. Coarser estimates did not adequately distinguish route assignment costs, increasing the net ordering dependence of the algorithm.

3.3 Pseudo-pin Assignment

The pseudo-pin assignment problem is the determination of net crossing locations along subproblem boundaries. Net boundary crossings must be consistent between sub-

problems to allow generation of the entire solution from final subproblem solutions. These constraints are introduced by creating a pair of pins to represent each net crossing. Each pin belongs to the appropriate subproblem on either side of the corresponding boundary and is referred to as a *pseudo-pin* because it is not part of the original problem specification. The consistency constraints are enforced by guaranteeing that the pseudo-pins of each pair have the same position with respect to the axis of the corresponding boundary. Initially, pseudo-pin positions are known only as the range corresponding to their assigned boundary. For the final layout solution, this range must be restricted to correspond to the final layout grid. Several choices exist for performing this refinement. Location decisions may be based on local or global information and positions may be restricted gradually or all at once. These choices and possible algorithms are introduced in the following. Algorithm comparisons are presented in Section 5.1.2.

The extreme case of determining pseudo-pin locations based on local information corresponds to solving the pseudo-pin problem during subsequent subproblem solution. Pseudo-pins may be treated as cells and assigning a pseudo-pin cell to a vertex of the 2×2 grid-graph corresponds to restricting the pseudo-pin's position to a portion of its original range. Unfortunately, this approach makes the final layout solution dependent on subproblem solution order because of the consistency constraints on pseudo-pin pairs. This is an undesirable side-effect because the best subproblem solution order is not known and it is possible that all possible solution orders will lead to infeasible subproblems. At best, subproblems early in the order will be optimized at the expense of subproblems later in the order.

Alternately, an algorithm can use more global information to determine pseudo-pin positions. In this paradigm, pseudo-pin pair locations are computed taking into account features of subproblems on both sides of the boundary. An advantage of this method is its potential for making the final solution independent of subproblem solution order. The minimum requirement for this independence is that at each level of the hierarchy pseudo-pin positions must be refined enough to correspond to single grids on the next level of subproblems. One approach to meeting this requirement is to solve 2×2 abstractions that divide each pseudo-pin range into the appropriate sections. This method was proposed by Brouwer and Banerjee [6]. Pseudo-pin locations are determined by solving a tree of recursive 2×2 abstractions along the internal axes of each routing subproblem. Figure 3.8 shows how the routing region is decomposed recursively along the axes of interest to completely determine

pseudo-pin positions. After a given 2×2 abstraction is solved, pseudo-pin locations are specified on its internal boundaries by first recursive subdivision and solution along the Y-axis down to the final grid level and then similar recursive subdivision and solution along the X-axis.

The other pseudo-pin assignment choice involves the amount of refinement performed at each opportunity. The method of Brouwer and Banerjee described above assigns pseudo-pins to final grid locations at the first opportunity. Instead of this immediate pseudo-pin assignment, another possibility is to perform a more gradual refinement of pseudo-pin locations. At each level of the problem hierarchy, pseudo-pin positions can be refined just enough to make the final solution independent of the subproblem solution order on the next level of hierarchy. This method solves the same abstractions as the immediate assignment approach, but the abstractions are solved at different points during the solution process. Figure 3.9 shows the sequence of subproblem abstractions that are solved when this technique is applied to the example of Figure 3.8. The solution of the more detailed pseudo-pin assignment abstractions is postponed to later in the overall solution process.

3.4 Rip-up and Re-route

Since the routing problem and by extension the layout problem is NP complete [18], any constructive heuristic algorithm used for solution of these problems can not guarantee success on all examples and a mechanism for backtracking and changing poor decisions can be invaluable. A common example of this concept is the rip-up and re-routing of nets [13, 26, 35, 44]. For this layout system, a simple re-route capability has been implemented using the 2×2 route assignment algorithm.

The re-route procedure is based on re-routing nets within a sliding 2×2 window. At each level of the hierarchy, any cluster of four routing cells which form a 2×2 abstraction can be selected and the net segments passing through them re-routed. This selection only occurs if subproblems corresponding to the current level can not be successfully routed and an abstraction is chosen only if it is possible to reduce capacity violations on a particular edge of the abstraction by re-routing nets onto adjacent edges with extra capacity. More sophisticated selection of sliding 2×2 windows is planned for the future. Table 3.4 compares results and program execution times for the separate phase routing algorithm with and without the re-route option. The examples and metrics are the same as those used for

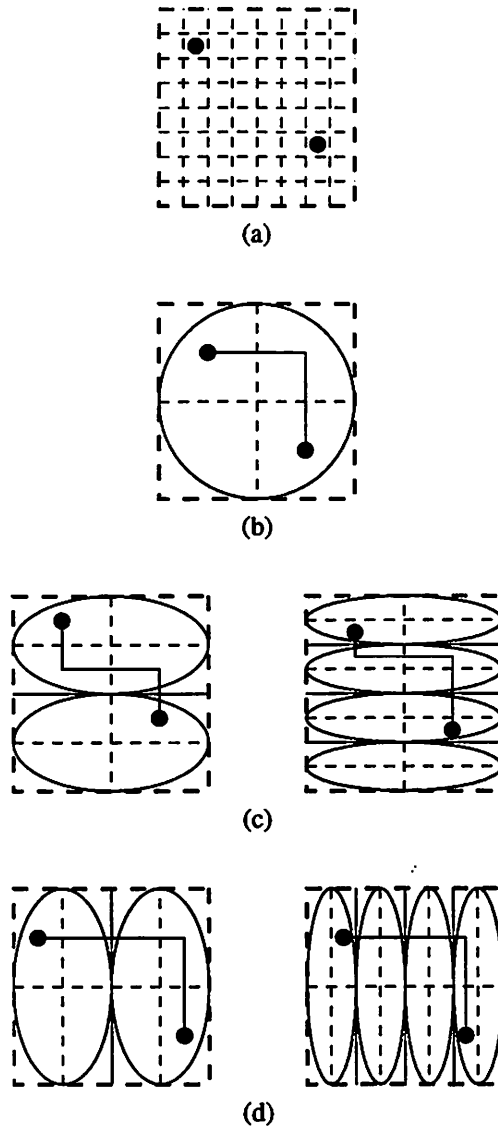
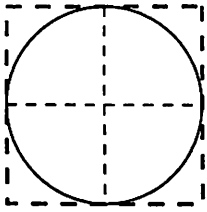
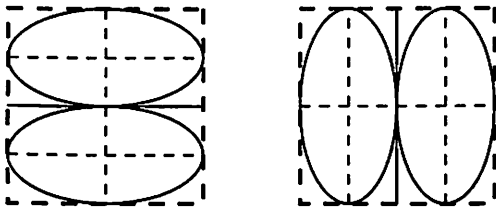


Figure 3.8: Immediate pseudo-pin assignment. (a) original routing problem (b) initial 2×2 abstraction (c) subproblem abstraction sequence for pseudo-pin assignment along Y-axis (d) subproblem abstraction sequence for pseudo-pin assignment along X-axis

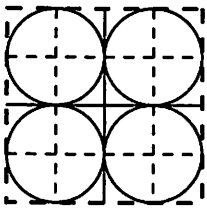
Level 0: Initial Subproblem Abstraction



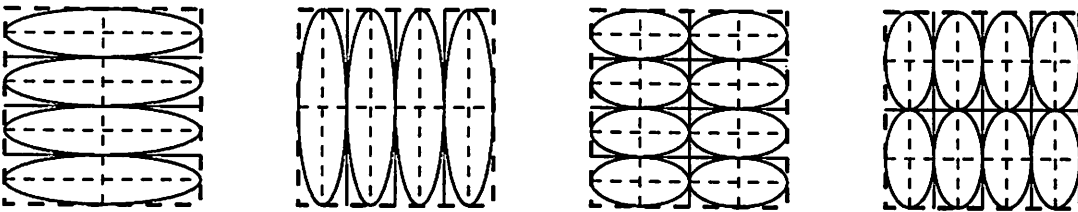
Level 0: Pseudo-pin Assignment Abstractions



Level 1: Subproblem Abstractions



Level 1: Pseudo-pin Assignment Abstractions



Level 2: Final Subproblem Abstractions

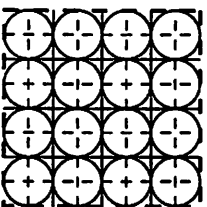


Figure 3.9: Gradual pseudo-pin assignment

Example	Net Count	Capacity Violations		Execution Time, (s)	
		No Re-route	Re-route	No Re-route	Re-route
misex3c	309	0	0	90	86
C880	317	5	3	70	73
duke2	321	0	0	82	83
C1355	333	9	9	71	68
misex3	358	0	0	95	97
C1908	359	6	5	82	80
C3540	834	46	43	248	265
C5315	1251	1106	1101	495	539
C7552	1572	589	584	647	650
C6288	2417	13	10	1153	1128

Table 3.4: Effects of applying a simple re-route procedure

Table 3.2. Despite its limitations and simplicity, the re-route capability is still beneficial. Though the reductions in capacity violations are limited, the increase in execution time is small or negligible. A more aggressive use of the sliding windows should produce larger improvements. In particular, no attempt is made currently to remedy capacity violations that occur in pseudo-pin assignment problems. Also, in general, the sliding window technique can be used to relieve and redistribute routing congestion before any actual edge capacity violations become evident.

Chapter 4

Separate Placement and Routing

The effectiveness of different methods for combining placement and routing will be measured by comparison with the standard layout paradigm of separate sequential placement and routing phases. The separate phase paradigm has been implemented with layout phases that are based on the same basic 2×2 layout algorithms to provide consistent comparisons between the methods. Ideally, the separate phase algorithms should be the best possible so that any improvement found using a combined approach can be attributed to the combination rather than poor separate phase solutions. The usefulness of the separate phase implementation as a reference is evaluated by comparing it with the TimberWolf Version 6.1 layout system [32]. The TimberWolf placement and routing package is a widely available layout system that is used frequently for benchmark comparisons. The results generated by TimberWolf may be considered to represent currently attainable layout solutions.

4.1 Placement

The separate placement phase algorithm combines hierarchical decomposition and the 2×2 cell assignment algorithm. Each reduction step is based on the solution of a 2×2 placement abstraction and solving the hierarchy of placement subproblems results in a complete placement solution. The process differs from the basic methods described in Chapter 3 in that global placement information is used to create each subproblem abstraction instead of just parent abstraction information and in that the resulting subproblem solution interdependencies are attenuated by iterating the solution of subproblems within each level. These differences are discussed in the following along with a more detailed description of

constraint and objective function calculations for the 2×2 placement abstraction.

Global information about external cell locations is propagated into problem abstractions based on a method introduced by Dunlop and Kernighan [14] called *terminal propagation*. They report that using this enhancement reduces the number of required routing tracks by as much as 30%. The method uses *dummy* cells to represent the global information. A cell of zero weight is created for each cell outside a problem region that is connected to a cell inside the region. The set of legal placement locations for the cell is restricted to represent the direction of the external connection. The rules used in this research for constructing dummy cell placement constraints are illustrated in Figure 4.1. The region surrounding a problem abstraction is divided into areas based on the size of the problem region. External connection location is based on the location of the terminal of the external cell that is connected to a cell of the problem. Each area is labeled with a placement condition that holds for the dummy cell if the closest edge of the corresponding external terminal's bounding box intersects the area.

Unfortunately, using this external cell information creates dependencies between the solution of a subproblem and the solutions of other subproblems on its level of the hierarchy. Since the locations of external connections affect the constraints introduced to each abstraction, the placement solution is dependent on the order of subproblem solution within a level of the hierarchy. To reduce the effects of this dependency, the solution of all subproblems in a level of hierarchy is iterated until combined solution improvement ceases. Suaris and Kedem report that this iterative procedure improves the global minimal cut weight by up to 10% and that the improvement is achieved in two to three passes.

The 2×2 cell assignment constraint values and objective function components are determined as follows. Area targets for each partition block are set as

$$target(\mathcal{M}_{v_j}) = \sum_{k=1}^M weight(m_k) \times \frac{limit(v_j)}{\sum_{v_i \in \mathcal{V}} limit(v_i)}.$$

This implements the goal of distributing cells in proportion to the size of the layout region corresponding to each partition. The maximum error, ϵ , is set to allow the cell distribution to be out of proportion by at most one cell. The net weight functions are made to implement

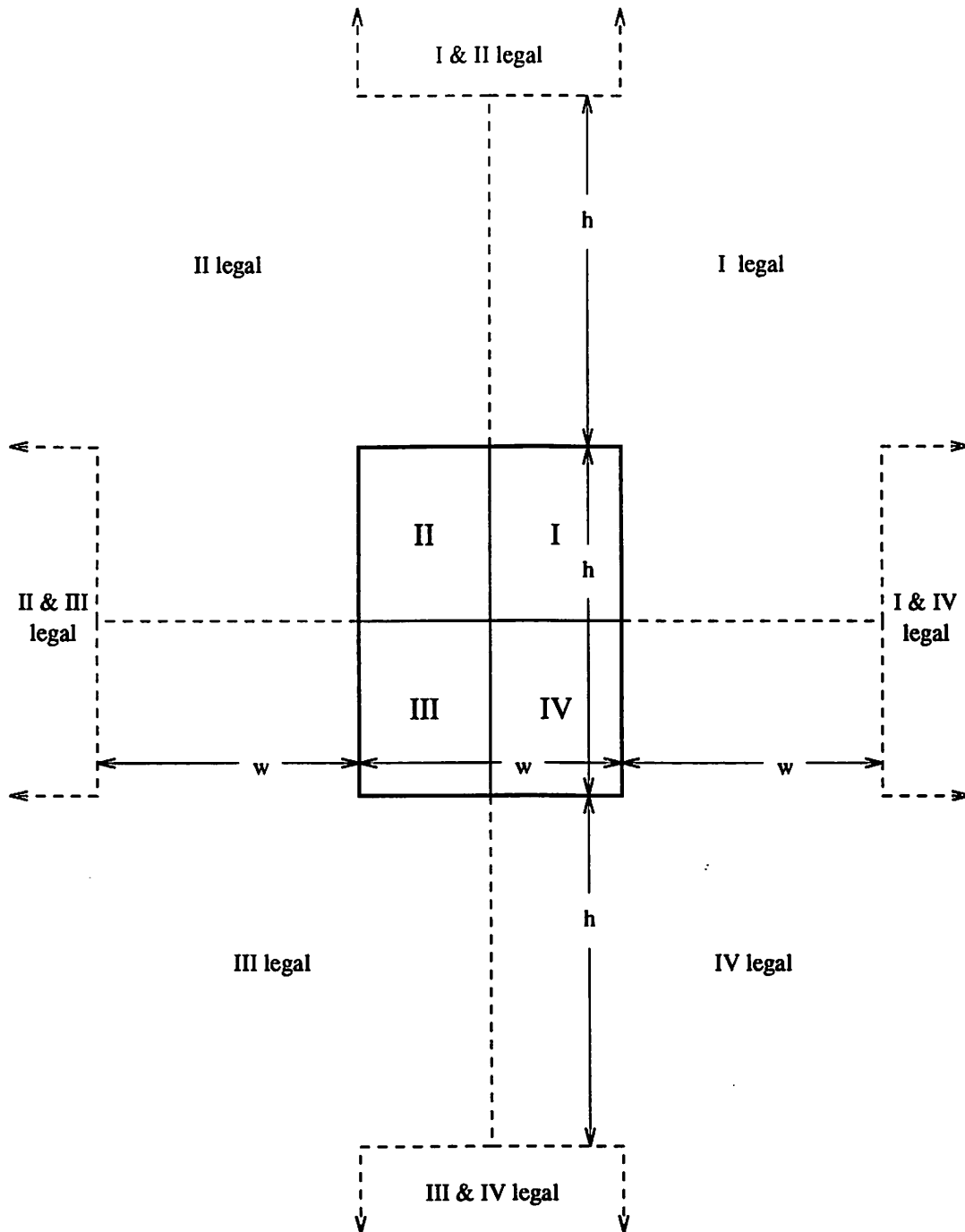


Figure 4.1: Placement constraints for terminal propagation

a typical net crossing metric by setting

$$\begin{aligned} \text{weight}(e_k) &= 2, \quad k = 1, 3, \\ \text{weight}(e_k) &= 1, \quad k = 2, 4, \\ \text{weight}(v_j) &= 0, \quad \forall j, \end{aligned}$$

and

$$C_1 = C_2 = 1.$$

The edges that cross horizontal partition boundaries are assigned weights twice that of edges that cross vertical partition boundaries to reflect the expected routing capacity distribution.

4.2 Routing

The separate global routing phase algorithm uses hierarchical decomposition and the 2×2 route assignment algorithms. Routing is performed using the same hierarchy as the placement phase and the placement grid is equivalent to the detailed global routing grid, so performing route assignment and pseudo-pin assignment on the subproblems of the placement hierarchy results in a complete global routing. Pseudo-pin assignment during routing uses the immediate assignment method described in Section 3.3. The grid model for routing capacity constraints and the heuristic capacity calculations are described in the following.

The region corresponding to a 2×2 abstraction is an array of placement grids. Associated with each placement grid location is a value for the number of feed-throughs and the number of routing tracks that may pass through that region. If a cell has been placed at a grid, then these values are determined by the number of feed-throughs and tracks that may pass through or over the cell. Otherwise, the values are the maximum possible. The number of tracks available for routing across a vertical boundary between placement locations is considered to be the minimum of the number of available routing tracks in two adjacent locations. Similarly, the number of feed-throughs that may cross a horizontal boundary is the minimum of the number of feed-throughs that may pass through the adjacent grid locations. The array of placement grids corresponds to a portion of the detailed global routing graph. Each placement grid corresponds to a vertex and each boundary between placement grids corresponds to an edge. The capacity of each edge is the number of routes that may cross the corresponding boundary.

(n,1)	...	(n,p)	(n,q)	...	(n,m)
.
.
.
(j,1)	...	(j,p)	(j,q)	...	(j,m)
(i,1)	...	(i,p)	(i,q)	...	(i,m)
.
.
.
(1,1)	...	(1,p)	(1,q)	...	(1,m)

Figure 4.2: Subproblem abstraction on a placement grid

Suppose that the subproblem boundaries are between columns p and q and rows i and j . This situation is illustrated in Figure 4.2. The capacity of edge e_2 of the 2×2 abstraction is the number of routes that may cross the placement grid boundaries between columns p and q over the range from row j to row n inclusive. If routing capacity distribution is nonuniform, considering only the capacity of the placement boundaries corresponding to the subproblem boundaries may cause routing problems. For example, if adjacent placement grid boundaries have smaller routing capacities, an overly optimistic number of routes may be assigned to cross the current subproblem boundary, resulting in subsequent capacity violations on the adjacent boundaries. To avoid some of these problems, a heuristic proposed

by Burstein and Pelavin [10] is applied to calculate capacities. The calculations for edge e_2 are presented to illustrate the method. Analogous conditions and calculations hold for the other edges. Let the placement grid at row r and column c be

$$g(r, c)$$

and let the routing capacity between grids $g(r, c)$ and $g(r, c + 1)$ be

$$capacity(r, c).$$

Calculation of the capacity of edge e_2 first requires determining modified capacities

$$capacity^*(r, c) \quad r = i, \dots, n, \quad c = 1, \dots, m - 1$$

as follows:

$$\begin{aligned} capacity^*(r, p) &= capacity(r, p), \\ capacity^*(r, c) &= \min\{capacity(r, c), capacity^*(r, c + 1)\} \quad c < p, \\ capacity^*(r, c) &= \min\{capacity(r, c), capacity^*(r, c - 1)\} \quad c > p. \end{aligned}$$

Now, the capacity of edge e_2 is calculated as

$$\sum_{r=j}^n \frac{\sum_{c=1}^{m-1} capacity^*(r, c)}{m - 1}.$$

Via capacities are calculated using

$$via_capacity(v_k) = v(capacity(e_i), capacity(e_j)) \times C(n)$$

where e_i and e_j are the edges adjacent to v_k , n is the number of placement locations in the region corresponding to v_k , $v(a, b)$ is the minimal via capacity computed in [25], and

$$C(x) = \frac{2x}{x + 1}.$$

4.3 Results

The layout solutions of the 2×2 separate phase approach were compared with solutions generated by TimberWolf 6.1 to measure the performance of the basic 2×2 algorithms. Whenever possible, both systems were given the same layout constraints and results were generated as would be required in production practice. The goal was to use

each system to find a minimal area layout for each example that admits a feasible global routing of minimal length. Standard net length and area metrics are used to compare the results generated by each method.

The 2×2 separate phase approach uses pad positions and a layout area boundary definition as problem constraints in addition to the basic circuit description. Pad positions are specified as fractional distances along specific chip edges. For all the examples, these positions were generated using the *Octtools*[16] `padplace` program. Since the hierarchical decomposition is confined to the given layout region, many algorithm executions on regions of different sizes and shapes may be required to find a feasible solution. This search was performed by hand with a secondary goal of unity aspect ratio.

The TimberWolf placement and routing package is a widely available layout system that is used frequently for benchmark comparisons. The system handles gate-array designs by treating them as standard cell problems and its primary goal is to minimize layout solution area for a given aspect ratio. For sea-of-gates designs, an additional post-processing phase is required to translate the standard cell solution into a legal sea-of-gates placement.

Results for the TimberWolf system are based on its gate-array solution of the circuit examples given an aspect ratio goal of unity and the same pad position specifications as used for the separate phase 2×2 algorithms. Since the gate-array problem is really solved as a standard cell problem, each cell template in the sea-of-gates library was represented as a standard cell of the same dimensions and with the same number of routing feed-throughs and free routing tracks. Horizontal routing segments were favored over vertical routing segments by a factor of two to match the ratio of horizontal to vertical routing capacity in a typical library cell.

The TimberWolf results require post-processing to generate legal sea-of-gates placements because the TimberWolf system is not capable of handling certain sea-of-gates design constraints. Though TimberWolf can be constrained to place cells on grid locations within cell rows, it can not be coerced to space cell rows on placement grids. Also, the current sea-of-gates array requires cells in odd-numbered columns to be placed mirrored about the vertical axis. While TimberWolf has an optimization step that mirrors cells to improve the layout solution, it can not handle mirroring cells based on their final location. The post-processing step maps cells to sea-of-gates grids by placing cell rows on placement grid locations while spacing the rows far enough apart to accommodate the appropriate maximum channel densities specified in the TimberWolf output. Admissible odd column cell

Example	Semiperimeter Net Length (λ)		
	TimberWolf	Separate	Separate TimberWolf
C880	237956	177620	0.75
C1355	216006	194907	0.90
misex3c	242499	163422	0.67
duke2	290811	186062	0.64
C1908	264973	208189	0.79
misex3	276730	195043	0.70
C3540	907437	626381	0.69
C5315	2127501	2335723	1.10
C7552	2441560	1945577	0.80
C6288	1878856	1441928	0.77

Table 4.1: Total net length comparisons between the separate phase 2×2 algorithms and TimberWolf 6.1

mirroring was obtained by turning off the cell mirroring optimization step during TimberWolf execution and then mirroring all cells in odd columns during the post-process mapping step. Unfortunately, the post-processing changes the relative positions of net terminals and routing feasibility of the subsequent result can not be guaranteed. However, since the spacing between cell rows is generally increased, the likelihood that the channel routing problems will become infeasible is small and the post-processed TimberWolf results represent reasonable approximations of currently attainable layout solutions.

Comparisons of the separate phase versions of the basic 2×2 algorithms and TimberWolf 6.1 are presented in Table 4.1 and Table 4.2. Total net length and layout area are the metrics used. The length of a net is measured as one-half the perimeter of its bounding box and layout area is measured as the bounding box of the core cell area. The separate phase algorithms performed better than TimberWolf 6.1 on all examples except for example C5315. The total net length obtained by the separate phase algorithms varies from 36% less to 10% more than the TimberWolf total net length results and the layout area varies from 69% less to 54% more than the TimberWolf area. Example C5315 is a particularly difficult example for the separate phase algorithms given fixed layout area constraints. This example contains many dense, tightly coupled subcircuits and the placement algorithm places them close together as desired. Unfortunately, the routing algorithm's predictions of routing capacity are less accurate in the presence of such highly localized congestion

Example	Aspect Ratio (h/w)		Layout Area (λ^2)		
	TimberWolf	Separate	TimberWolf	Separate	$\frac{\text{Separate}}{\text{TimberWolf}}$
C880	$29/26 = 1.12$	$17/19 = 0.89$	5838222	2500989	0.43
C1355	$27/27 = 1.00$	$17/24 = 0.71$	5644647	3159144	0.56
misex3c	$36/27 = 1.33$	$17/18 = 0.94$	7526196	2369358	0.31
duke2	$35/28 = 1.25$	$18/17 = 1.06$	7588140	2369358	0.31
C1908	$31/28 = 1.11$	$18/24 = 0.75$	6720924	3344976	0.50
misex3	$36/29 = 1.24$	$19/19 = 1.00$	8083692	2795223	0.35
C3540	$62/42 = 1.48$	$32/35 = 0.91$	20162772	8672160	0.43
C5315	$70/49 = 1.43$	$66/80 = 0.83$	26558490	40883040	1.54
C7552	$76/55 = 1.38$	$55/69 = 0.80$	32365740	29384685	0.91
C6288	$101/73 = 1.38$	$48/51 = 0.94$	57089139	18954864	0.33

Table 4.2: Area and aspect ratio comparisons between the separate phase 2×2 algorithms and TimberWolf 6.1

and sometimes too many feed-through routes are assigned at the higher levels of routing abstraction than can actually be wired, resulting in capacity violations on the final routing grid. Overall, the results for the separate phase algorithms compare well with the results for TimberWolf 6.1 and this performance indicates that using the separate phase algorithms as reference cases will provide useful measurements of the utility of different methods of combining placement and routing.

Chapter 5

Placement and Routing Communication

Improved communication between phases is an important element in integrating placement and routing to obtain better layout solutions. A fundamental premise for combining placement and routing is that information derived in each phase is required by the other to make better decisions. Routing is clearly dependent on placement because formulation of the routing problem requires the placement solution. The interesting aspect of this relationship is how much the routing depends on the accuracy of the placement information, especially since the most obvious methods of solving placement and routing concurrently require the routing algorithm to use coarse or approximate placement data. Conversely, placement is not so much dependent on routing as it is validated by it. The real goal of the placement phase is to provide for a successful routing phase and feedback from the routing algorithm should be useful for guiding placement decisions. If the layout process is viewed as a search for a feasible solution, then the feedback can be viewed as an intelligent method for guiding this search. In this context, the hierarchy provides structure to the search and for the information flow and the common data model provides a common reference frame for efficient information transfer.

5.1 Routing

The routing problem is to determine a minimal cost set of interconnection patterns for a given circuit net list subject to particular routing problem constraints. The relevant

constraints for this discussion are net terminal locations and capacity restrictions. The wiring patterns must implement nets by connecting terminals at particular locations and the collection of patterns must be selected so that a working circuit can be manufactured. Decisions on interconnections are affected by the quality and granularity of data for terminal locations and capacity values. These data are determined by both placement and intermediate routing decisions.

5.1.1 Placement Resolution

Routing depends directly on placement because the placement solution specifies the routing problem. Since solving placement and routing concurrently requires making routing decisions based on incomplete or approximate placement solutions, any attempt at combining placement and routing should account for the dependency of the routing portion of the approach on the granularity of the available placement information. This dependence can be measured for the hierarchical decomposition approach by varying the number of more detailed placement subproblem levels that are solved before solving a given level of routing subproblems.

The major features of a routing problem, pin locations and routing capacity distribution, are generated directly from the placement solution. Each cell has pins which must be connected to pins on other cells. Each cell also has internal wiring patterns which present obstacles to subsequent routing. Thus, cell positions specify net terminal locations and determine the distribution of capacity throughout the routing region. The accuracy of both terminal position data and capacity values is limited by the granularity of the cell placement information.

The problem hierarchy generated by the hierarchical decomposition can be used to explore the dependence of routing algorithm performance on placement information resolution. Since each level of the hierarchical decomposition method represents a further refinement of the solution and since both placement and routing use the same problem reduction hierarchy, a level of the hierarchy is a convenient unit for measuring placement data granularity. Routing dependence on placement granularity is examined by varying the number of more detailed levels of placement that are solved before routing a given level. The number of levels can vary from zero to the total number of levels in the hierarchy. During solution, cells are assigned to partition blocks as in the separate 2×2 placement

phase, using the 2×2 cell assignment algorithm along with terminal propagation and iterative improvement within each level. However, the terminal propagation is limited to nets defined by the most detailed routing subproblems available. Routes and pseudo-pin positions are assigned based on the current placement resolution.

The sequences of cell and route assignment abstractions solved for various levels of more detailed placement are shown in Figures 5.1, 5.2, and 5.3. Route assignment is delayed by the specified number of levels subject to fundamental abstraction constraints. For each 2×2 abstraction, the route assignment requires resolution of each terminal to within a specific grid of the abstraction. This applies to both routing and pseudo-pin assignment abstractions. Routing subproblems require placement solutions commensurate with the current grid abstraction and the subsequent pseudo-pin assignment subproblems require at least one more level of detailed placement information. At each level, pseudo-pins are refined as much as is possible given the current granularity of both placement and routing solutions.

Figure 5.1 shows the sequence for assigning routes on a 2×2 abstraction before cell assignment is solved on any more detailed levels of the hierarchy and corresponds to the most parallel execution of placement and routing. The route assignment is possible because the minimum resolution requirement on net terminal positions is that the terminals be classifiable with respect to the 2×2 abstraction. This requirement is met after solving the placement problem on the abstraction and represents assigning routes based on the most approximate placement data possible.

Figure 5.2 illustrates the subproblem sequence for solving one more level of placement subproblems before routing a given level of subproblems. This situation differs from the zero level case in that the routing subproblems are based on more refined placement data. The pseudo-pin assignment subproblems are still based on the same level of placement refinement, but the overall ordering of subproblem solutions is slightly different because of the delay in route assignment. For all of the cases based on an intermediate number of more detailed placement levels, initial pseudo-pin assignment is performed immediately after each level of routing subproblems and pseudo-pin locations are refined to a level of detail commensurate with the current level of placement granularity. Pseudo-pin positions are further refined after each subsequent level of placement solution. This approach to pseudo-pin assignment provides the routing algorithm with as much detail as the current level of placement resolution allows.

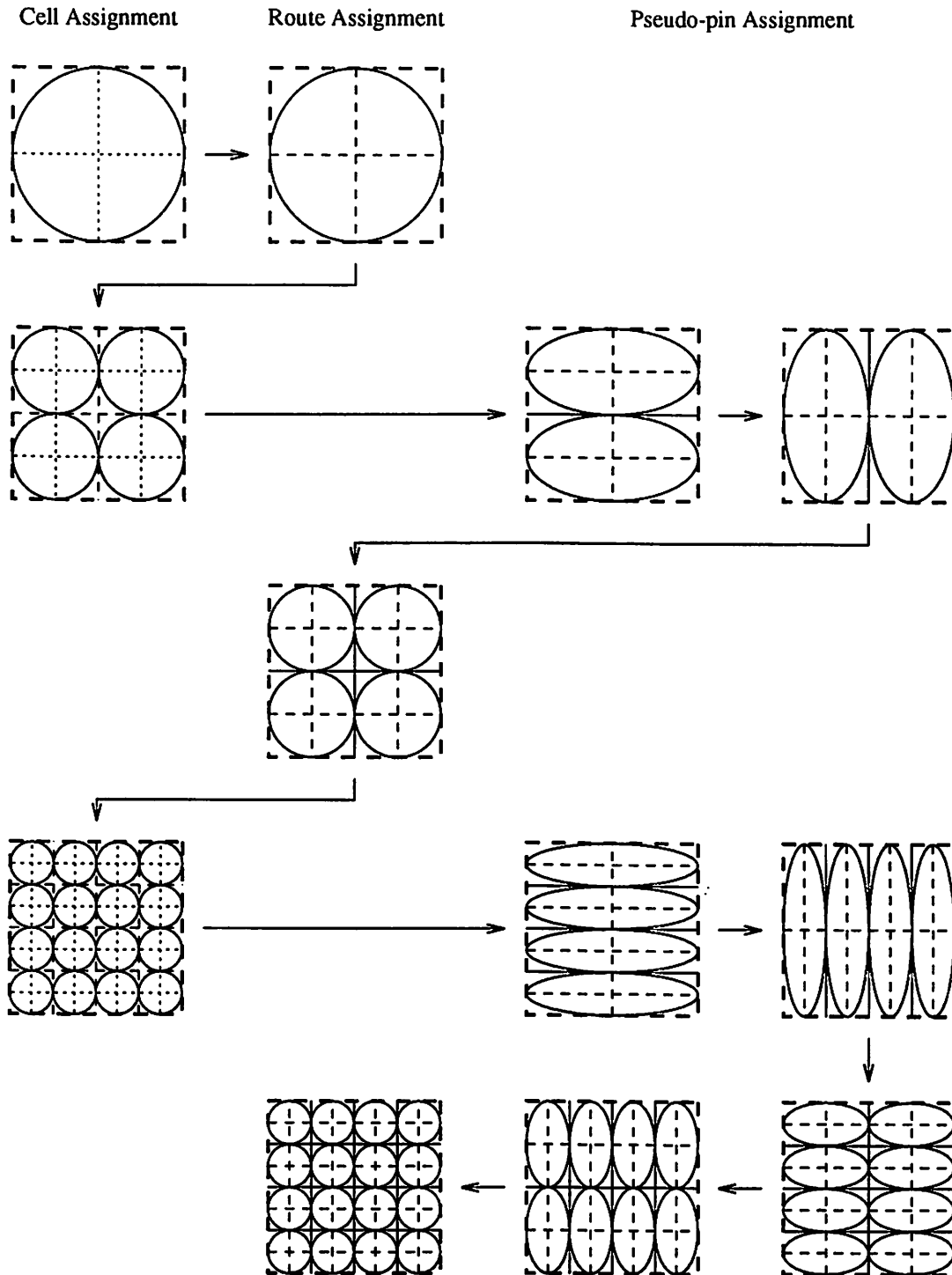


Figure 5.1: Subproblem abstraction sequence for solving a routing level immediately after the corresponding placement level

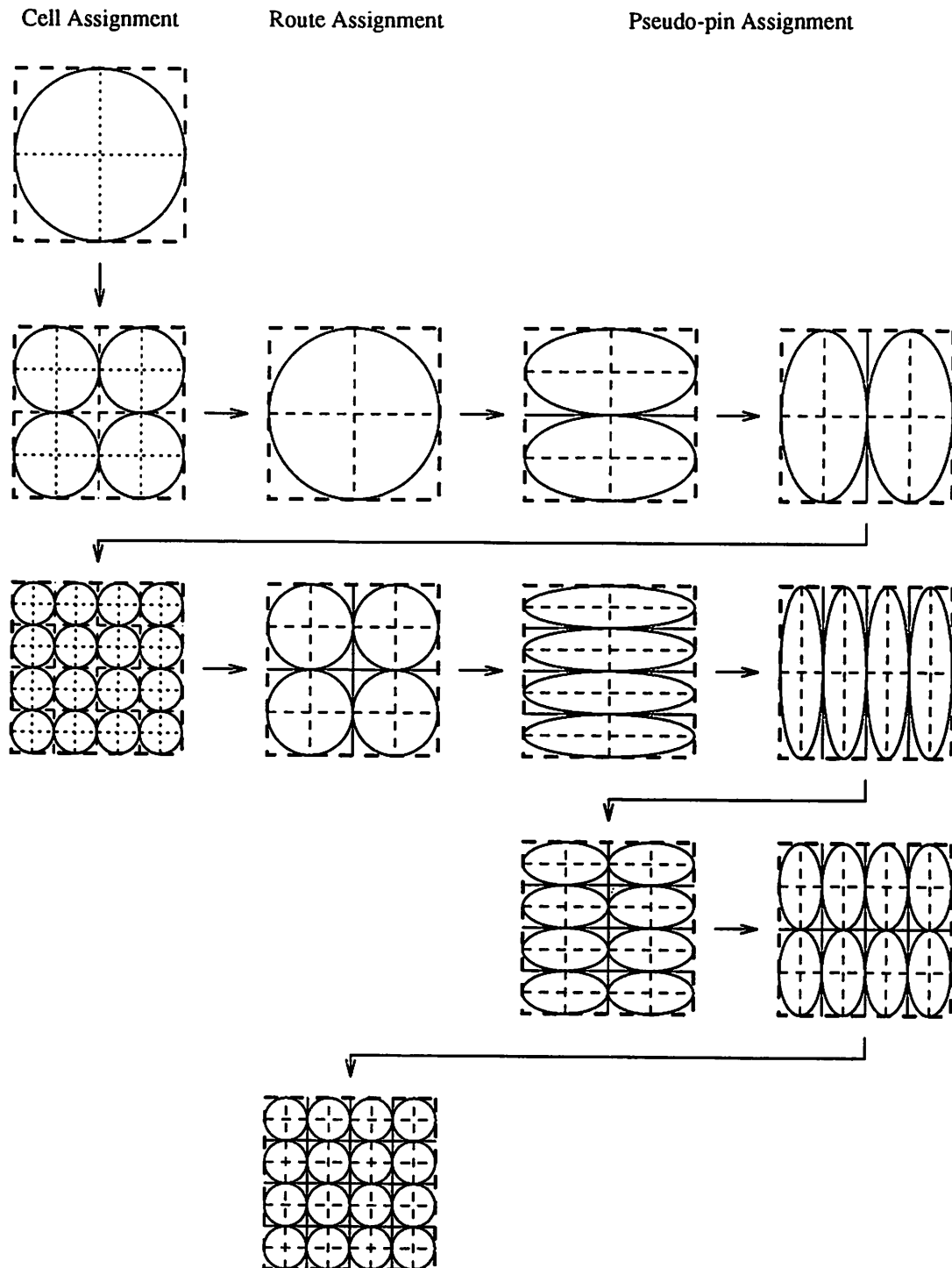


Figure 5.2: Subproblem abstraction sequence for solving one more level of placement before routing

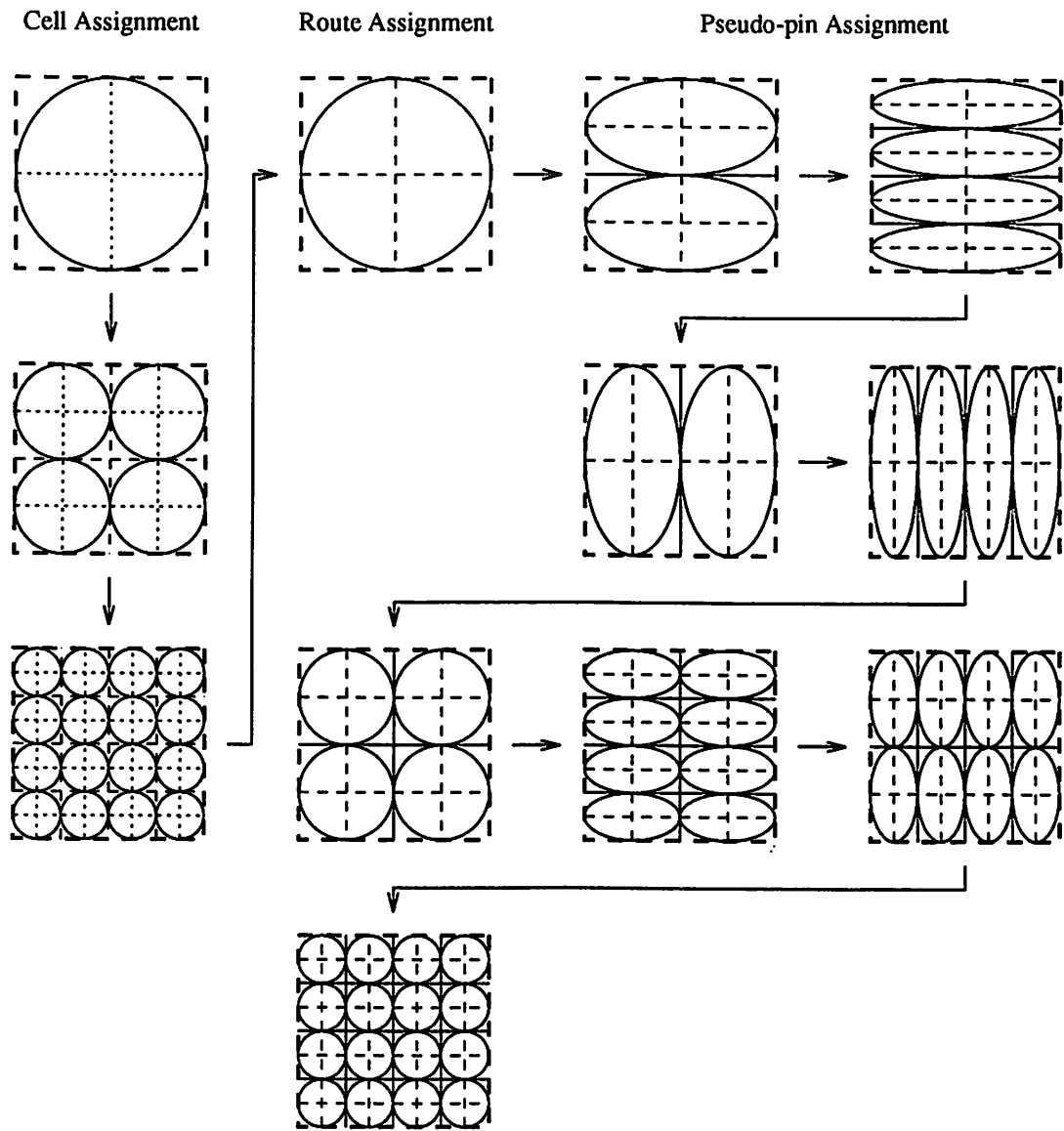


Figure 5.3: Subproblem abstraction sequence for solving all levels of placement before routing

Figure 5.3 shows solving the complete hierarchy of placement subproblems before solving the first level of routing and corresponds to performing separate sequential phases of placement and routing. Since the final cell positions are known exactly, routes are assigned based on the most exact placement data possible.

Routing and pseudo-pin assignment subproblems are solved using the 2×2 route assignment algorithm. The components of this algorithm that are affected by placement granularity are the net length and capacity calculations.

The 2×2 route assignment algorithm uses net terminal locations to classify nets into types and to estimate net length. If cell positions are known exactly, then the exact terminal locations are used. Otherwise, the computations assume that terminals are located at the center of the range determined by the current placement granularity.

Routing capacity values are derived as accurately as possible given the current level of placement detail. The capacity heuristics assume an array or grid of routing cells and the heuristics are applied to the routing super-grid corresponding to the most detailed level of completed placement hierarchy. For example, in Figure 5.2, the first routing subproblem uses capacity values derived from a 4×4 super-grid, corresponding to two levels of placement solution. Each placement subproblem on a level is a node in a routing super-grid-graph and the boundaries between the placement subproblems are edges in the graph. Since each grid location in the super-grid may correspond to several individual placement locations, the number of feed-throughs and tracks available for routing through and over a super-grid location must be estimated. The estimates are based on the cells assigned to the placement subproblem corresponding to the super-grid location. Each cell has a predetermined number of feed-throughs and tracks available for routing over or through the cell. Capacity estimates for a super-grid node v_k are calculated assuming that a cell with average values for its number of available feed-throughs and tracks is placed at every placement location in v_k and that routing resources are uniformly available across the region of v_k . The calculations count empty placement grids as cells with all feed-throughs and tracks available. Let:

$$\begin{aligned}
 \textit{feed_through}_{m_i} &= \text{the number of usable routing columns through cell } m_i, \\
 \textit{tracks}_{m_i} &= \text{the number of usable routing rows over cell } m_i, \\
 \textit{feed_through}_{grid} &= \text{the number of routing columns in a placement grid,} \\
 \textit{tracks}_{grid} &= \text{the number of routing rows in a placement grid,} \\
 \textit{empty_grids} &= \text{the number of empty placement grids in } v_k.
 \end{aligned}$$

Example	Total Levels	Edge Capacity Violations					
		0 levels	1 level	2 levels	3 levels	4 levels	5 levels
C880	4	8	2	2	5	5	5
C1355	4	25	16	7	10	9	9
misex3c	4	0	0	0	0	0	0
duke2	4	3	0	0	0	0	0
C1908	4	22	9	8	5	6	6
misex3	4	1	2	0	0	0	0
C3540	4	140	76	41	46	46	46
C5315	5	1507	1209	1165	1111	1067	1106
C7552	5	1161	802	728	616	619	589
C6288	5	93	25	38	12	12	13

Table 5.1: Edge capacity violations as a function of the number of levels of cell assignment performed ahead of route assignment

If v_k corresponds to a placement grid region of h rows and w columns, then feed-through and track estimates are calculated as:

$$\begin{aligned}
 \text{empty_grids} &= (h \times w - |\mathcal{M}_{v_k}|), \\
 \text{feed_throughs} &= w \times \frac{\text{empty_grids} \times \text{feed_throughs}_{\text{grid}} + \sum_{m_i \in \mathcal{M}_{v_k}} \text{feed_throughs}_{m_i}}{h \times w}, \\
 \text{tracks} &= h \times \frac{\text{empty_grids} \times \text{tracks}_{\text{grid}} + \sum_{m_i \in \mathcal{M}_{v_k}} \text{tracks}_{m_i}}{h \times w}.
 \end{aligned}$$

Via capacities are calculated based on edge capacities as for the separate phase algorithms (Chapter 4).

The results for performing cell assignment different numbers of levels ahead of route assignment are presented in Table 5.1. The examples were placed and routed as described above on the same minimum area arrays used in Chapter 3. Routing performance as measured by the number of global routing edge capacity violations generally improves with increased number of levels of extra cell assignment. There is also a small tendency for performance to decrease as the number of extra cell assignment levels approaches the total number of levels in the hierarchy, but this characteristic can be attributed to the level of intermediate routing decision refinement rather than placement data granularity.

The major relationship exhibited is for routing performance to improve with more levels of cell assignment performed ahead of route assignment. This confirms that good routing decisions require detailed placement information. The amount of improvement with

each extra level of placement refinement varies between examples, but often the majority of the benefit occurs after one or two levels. The other interesting feature is that a difference of one level of extra placement often makes a large difference in the performance of the routing algorithm.

A minor trend displayed is for slightly degraded routing performance with more levels of cell assignment. This is attributed to the policy of immediately refining pseudo-pin positions as much as allowed by the current level of placement granularity and is verified in the next section. Refining pseudo-pin locations based on insufficient local routing information may result in more difficult routing subproblems and lead to increased edge capacity violations.

5.1.2 Routing Decision Resolution

Routing algorithm performance is also affected by intermediate routing decisions made during the course of the algorithm. These intermediate decisions determine net terminal locations in subsequent routing subproblems and also affect capacity to the degree that the algorithm tries to predict and avoid future wiring congestion. The decisions are transmitted to subsequent subproblems through the specification of pseudo-pin positions. The dependence of the 2×2 route assignment algorithm on the granularity of intermediate routing decisions is explored by comparing the immediate and gradual methods of performing pseudo-pin assignment described in Section 3.3.

Pseudo-pin locations may be refined gradually or specified completely. The option chosen determines the granularity of the intermediate routing decisions presented to subsequent subproblems. The immediate method of pseudo-pin assignment provides more detailed information to subsequent subproblems by making more specific routing decisions earlier. The gradual method of pseudo-pin assignment delays making the same routing decisions and only provides approximate information to subsequent subproblems. The cost of the immediate approach is more cell assignment work. Generation of detailed pseudo-pin positions using the immediate approach requires a complete detailed placement. Also, the early decisions on pseudo-pin locations may be suboptimal or infeasible, leading to problems later in the decomposition process. The gradual approach is one attempt at addressing these issues. Only one extra level of placement hierarchy solution is required beyond the current routing level to perform the gradual pseudo-pin assignment and decisions on specific

Example	Edge Capacity Violations		Net Length (λ)	
	Immediate	Gradual	Immediate	Gradual
C880	5	4	—	—
C1355	9	8	—	—
misex3c	0	0	163422	163471
duke2	0	0	186062	186081
C1908	6	6	—	—
misex3	0	0	195043	195035
C3540	46	36	—	—
C5315	1106	1135	—	—
C7552	589	610	—	—
C6288	13	11	—	—

Table 5.2: Edge capacity violations and total net length for the immediate and gradual methods of pseudo-pin assignment

pseudo-pin locations are delayed until more local information is known.

Table 5.2 compares routing algorithm performance for the two different pseudo-pin assignment methods. Routing performance is measured by the number of final global routing graph edge capacity violations. If edge capacities were not violated for both cases, the total net length is compared. The length of a net is measured as one-half the perimeter of its bounding box. These results were obtained by applying the routing algorithm as a separate phase to the minimum area grid examples of Chapter 3. Delaying pseudo-pin decisions until more local information is known is often beneficial. Unfortunately, this effect is example or data dependent and is not always exhibited. However, the pseudo-pin assignment method comparisons are generally consistent with the placement granularity results shown in Table 5.1. In particular, these results are consistent with the conclusion that the degradation in routing performance with increased levels of cell assignment before route assignment is a function of the pseudo-pin assignments rather than the placement granularity. All the examples for which the gradual method of pseudo-pin assignment performed as well or better also exhibited a degradation in routing performance with more levels of cell assignment performed ahead of route assignment. Also, except for example **C5315**, the examples for which the immediate method of pseudo-pin assignment produced better results did not exhibit the degradation with more levels of cell assignment.

5.2 Placement

Routing feedback is the important communication issue for placement because a true placement evaluation is not possible without routing information. Standard heuristics like net length and minimum area work well, but the ultimate measure of placement quality is the feasibility of the subsequent routing problem and the optimality of the subsequent routing solution. While routing information could be used by a specialized algorithm to modify the placement, more can be learned about combining placement and routing by trying to provide routing information back to the cell assignment algorithm such that re-applying the algorithm to the given placement subproblem results in a better solution. Knowledge about placement and routing interaction must be applied to determine the appropriate feedback mechanism for improving placement performance on subsequent attempts. Unfortunately, this interaction is complex and not fully understood. For the 2×2 cell assignment algorithm, the possibilities explored for incorporating feedback are modifying existing constraints and adjusting algorithm parameters.

When revisiting a placement subproblem, the feedback mechanism can either prohibit previous choices or encourage selection of different alternatives. Though prohibiting certain selections is the more direct method of influencing decisions, identification of specific bad decisions can be difficult, especially if algorithmic failure to find a feasible solution results from a set or combination of decisions rather than a single choice. Attempts at avoiding a supposedly “bad” decision may lead to worse solutions. In particular, adding specific constraints to the placement problem based on routing results is not well understood. For example, prohibiting a particular cell from being assigned to a particular partition block is not necessarily very meaningful or useful. It is more probable that a certain set of cells should not be assigned to the same partition block or that a specific distribution of cells among blocks is poor. To avoid these problems, no specific assignments are prohibited. Instead, selection of alternate assignments is encouraged by changing problem constraints and modifying algorithm parameters.

Two different problem constraints can be modified for the 2×2 cell assignment algorithm. First, the general distribution of cells among the partition blocks can be influenced by modifying the target sizes for each block. For example, partition blocks which correspond to congested routing subregions may have their target sizes lowered. Second, the routing results can be used to generate the dummy cells for terminal propagation. Instead

of creating a dummy cell for each external cell connection, a dummy cell is created for each routing edge that crosses the placement region boundary. The dummy cell partition block membership constraints are based on the locations of their corresponding routing segments.

Routing information can also be used to modify the 2×2 cell assignment algorithm edge and vertex weight parameters. Edge weights may be assigned such that edges with higher utilization in the corresponding routing problem have proportionally higher values. Similarly, vertex weights may be assigned such that vertices corresponding to more highly congested routing subregions have proportionally higher values.

Feedback information is derived from both the placement and routing solutions. Relevant data includes capacity values, edge utilization, and subregion routing congestion. The capacity and edge utilization values come directly from the routing problem formulation and solution. Subregion routing congestion is measured heuristically. The congestion calculations, feedback implementation details, and evaluation of feedback performance are described and discussed in Chapter 6.

Chapter 6

Combining Placement and Routing

The hierarchical decomposition framework allows exploration of the range of placement and routing integration possibilities. Different points along this range are examined, taking into account the placement and routing dependencies discussed in Chapter 5. A unified approach to placement and routing is analyzed, but the problem still remains intractable. A merged approach to combining placement and routing is also tried in an attempt to provide more immediate feedback and closer coupling of the two phases. For this method to be successful, the benefits of the feedback and coupling must overcome the disadvantage of making routing decisions based on coarse placement information. Alternately, route assignment can be based on more refined placement data. A coupled approach is examined that makes routing decisions based on detailed placement information and performs re-placement based on every level of routing refinement. Finally, a hybrid approach is attempted to further investigate routing feedback to placement and to exploit a characteristic of the route assignment's dependence on placement information. Route assignment consists of two phases, net group assignment and individual net assignment. Only the individual net assignment is dependent on detailed placement information. The hybrid approach feeds back net group assignment information to placement and delays individual net assignment decisions until detailed placement is complete.

6.1 Unified Approach

The hierarchical decomposition technique can be used to provide a unified placement and routing approach to layout if the the unified 2×2 placement and routing problem can be solved. One possible formulation of this subproblem is presented in the following, but even on the simple 2×2 data model, the unified placement and routing problem is still too difficult. The unified 2×2 problem requires placing and routing a given circuit net list on the 2×2 grid-graph, $G(\mathcal{V}, \mathcal{E})$. Each cell must be assigned to one of the four grid-graph vertices and each net must be assigned a route configuration on the 2×2 grid-graph such that all assignments are admissible and consistent.

As described in Chapter 3, cell-to-vertex assignments are admissible if the set of cells assigned to a particular vertex does not exceed the vertex's limit and route-to-net assignments are admissible if all nets are routed and all routing resource constraints are satisfied. Route-to-net assignments are dependent on cell-to-vertex assignments because a net's abstract type is a function of the vertex assignments of its cells. Route assignments and cell assignments are consistent if each route connects the abstract net type defined by the cell assignments for its assigned net.

The 2×2 placement and routing problem can be stated as follows. Let all possible routes on G be the set

$$\mathcal{R} = \{r_1, \dots, r_R\}.$$

Given a circuit description of modules (or cells)

$$\mathcal{M} = \{m_1, \dots, m_M\}$$

and nets

$$\mathcal{N} = \{n_1, \dots, n_N\},$$

denote the set of cells connected by net n_i as \mathcal{M}_{n_i} . Define decision variables

$$x_{m_i, v_j} = \begin{cases} 1 & \text{if cell } m_i \text{ is assigned to vertex } v_j \\ 0 & \text{otherwise} \end{cases}$$

and

$$y_{n_i, r_j} = \begin{cases} 1 & \text{if net } n_i \text{ is assigned route configuration } r_j \\ 0 & \text{otherwise.} \end{cases}$$

Let:

$weight(m_i)$ = the weight of cell m_i ,

$limit(v_j)$ = the weight limit of vertex v_j ,

$capacity(e_k)$ = the capacity of edge e_k ,

$capacity(v_k)$ = the capacity of vertex v_k .

Optimize the 2×2 layout subject to

$$\begin{aligned} \sum_{j=1}^4 x_{m_i, v_j} &= 1, & i = 1, \dots, M, \\ \sum_{i=1}^M weight(m_i) \cdot x_{m_i, v_j} &\leq limit(v_j), & j = 1, \dots, 4, \\ \sum_{j=1}^R y_{n_i, r_j} &= 1, & i = 1, \dots, N, \\ \sum_{i=1}^N \sum_{\{j | e_k \in r_j\}} y_{n_i, r_j} &\leq capacity(e_k), & k = 1, \dots, 4, \\ \sum_{i=1}^N \sum_{\{j | v_k \in r_j\}} y_{n_i, r_j} &\leq capacity(v_k), & k = 1, \dots, 4, \end{aligned}$$

and

$$x_{m_i, v_j} + y_{n_k, r_l} \leq 1, \quad \forall (i, j, k, l) \in \mathcal{Q}$$

where

$$\mathcal{Q} = \{(i, j, k, l) \mid m_i \in \mathcal{M}_{n_k}, v_j \notin r_l, n_k \in \mathcal{N}, r_l \in \mathcal{R}\}.$$

The first $M + 4$ constraints in the definition are the placement admissibility constraints. The next $N + 8$ constraints are the routing admissibility constraints and the last set of constraints are the consistency constraints on the placement and routing assignments. The consistency constraints enforce the condition that if a cell is assigned to a particular vertex, then no net that is connected to the cell may be assigned a route configuration that does not include the vertex. The number of possible combinations of vertex choices and illegal route configuration selections corresponds to the number of vertices not covered by a route in Figure 3.5. Counting the four degenerate routes, R equals 16 for the 2×2 routing problem. Over these 16 configurations, the number of vertices left uncovered is 24. These combinations must be considered for each cell of each net, so an upper bound on the number of consistency constraints is $24MN$.

Unfortunately, using this problem formulation to solve the unified 2×2 placement and routing problem is impractical. The formulation is an integer programming problem that is linear in the size of the circuit. Current circuit designs are too large to allow effective solution of this integer program using current linear programming packages. Also, the integer program is difficult even on small circuits. A common practice is to relax the integer constraints and solve the resulting linear program to obtain approximate results. Unfortunately, relaxing the integer constraints on this particular formulation results in trivial non-integer solutions of little value, making both rounding methods and branch-and-bound algorithms ineffective.

6.2 Merged Approach

A merged approach to layout can also be implemented within the hierarchical decomposition framework. The merged or concurrent solution of placement and routing is an attempt to provide more immediate feedback from routing to placement. The two phases are merged as a result of solving both placement and routing on each 2×2 abstraction of the decomposition and pseudo-pin assignment hierarchies. This approach to layout more closely integrates placement and routing phases than the coupled paradigm which solves the placement and routing subproblem hierarchies separately, but provides less integration than a unified approach which would solve placement and routing simultaneously on each abstraction. The 2×2 cell and route assignment algorithms are applied sequentially to each subproblem abstraction, resulting in an interleaving of placement and routing functions over the course of the decomposition process. An approach similar to the gradual method of pseudo-pin assignment is used to refine pseudo-pin locations for the next level of subproblems and is the primary method of communicating routing information to the placement process. Feedback between placement and routing results from both the pseudo-pin assignment process and the use of pseudo-pin positions to implement a kind of terminal propagation.

The parameters of the 2×2 cell and route assignment algorithms are set to make the best use of the current level of problem information detail. The route assignment algorithm is set up as in Section 5.1.1 to calculate net terminal positions and capacity values based on the current level of placement detail. Since cell assignment is performed for all subproblem abstractions, the cell assignment algorithm parameters must be adapted

more closely to the characteristics of individual subproblems. To account for variations in the shape of subproblem regions, the net weight functions are set to:

$$\begin{aligned} weight(e_k) &= 100 \frac{\frac{1}{capacity(e_k)}}{\sum_{e_i \in \mathcal{E}} \frac{1}{capacity(e_i)}}, \quad \forall k, \\ weight(v_j) &= 0 \quad \forall j, \\ C_1 &= 1, \\ C_2 &= 1. \end{aligned}$$

The partition area targets and maximum error limit are set as before. Namely,

$$target(\mathcal{M}_{v_j}) = \sum_{k=1}^M weight(m_k) \times \frac{limit(v_j)}{\sum_{v_i \in \mathcal{V}} limit(v_i)}.$$

and $\epsilon = 1$.

The sequence of subproblem abstractions solved for the merged approach is illustrated in Figure 6.1. Since the pseudo-pin assignment abstractions overlap previous abstraction boundaries, performing both placement and routing on each abstraction provides a form of rip-up, re-placement and re-routing feedback. In addition to refining pseudo-pin locations, the solution of the pseudo-pin assignment abstractions also allows modification of previous global decisions based on more local information. For example, the solution to the initial abstraction specifies cell assignments to the four possible partition blocks. The first set of pseudo-pin assignment abstractions are designed to refine pseudo-pin locations along the Y-axis, corresponding to the vertical internal boundaries of the initial abstraction. Since the cell assignment algorithm is executed on these abstractions, a cell initially assigned to block 1 may be restricted to sub-blocks of block 1 or assigned to sub-blocks of block 2. Similarly, a cell initially assigned to block 2 may be restricted to sub-blocks of block 2 or assigned to sub-blocks of block 1. Analogous conditions apply to cells in blocks 3 and 4 and with respect to the pseudo-pin assignment abstractions along the X-axis. Assigning a cell to a different block rather than restricting its position within its current partition represents a re-placement of that cell and the subsequent route assignment provides the required re-routing. At more detailed levels of the hierarchy, cell assignment on the pseudo-pin assignment abstractions both restricts and relaxes cell position ranges. For example, the partitions of the first set of Y-axis pseudo-pin assignment abstractions for level 1 overlap multiple partitions of the level 1 subproblems. Any cell whose range of positions has been relaxed must have its range restricted again to allow formulation of the orthogonal

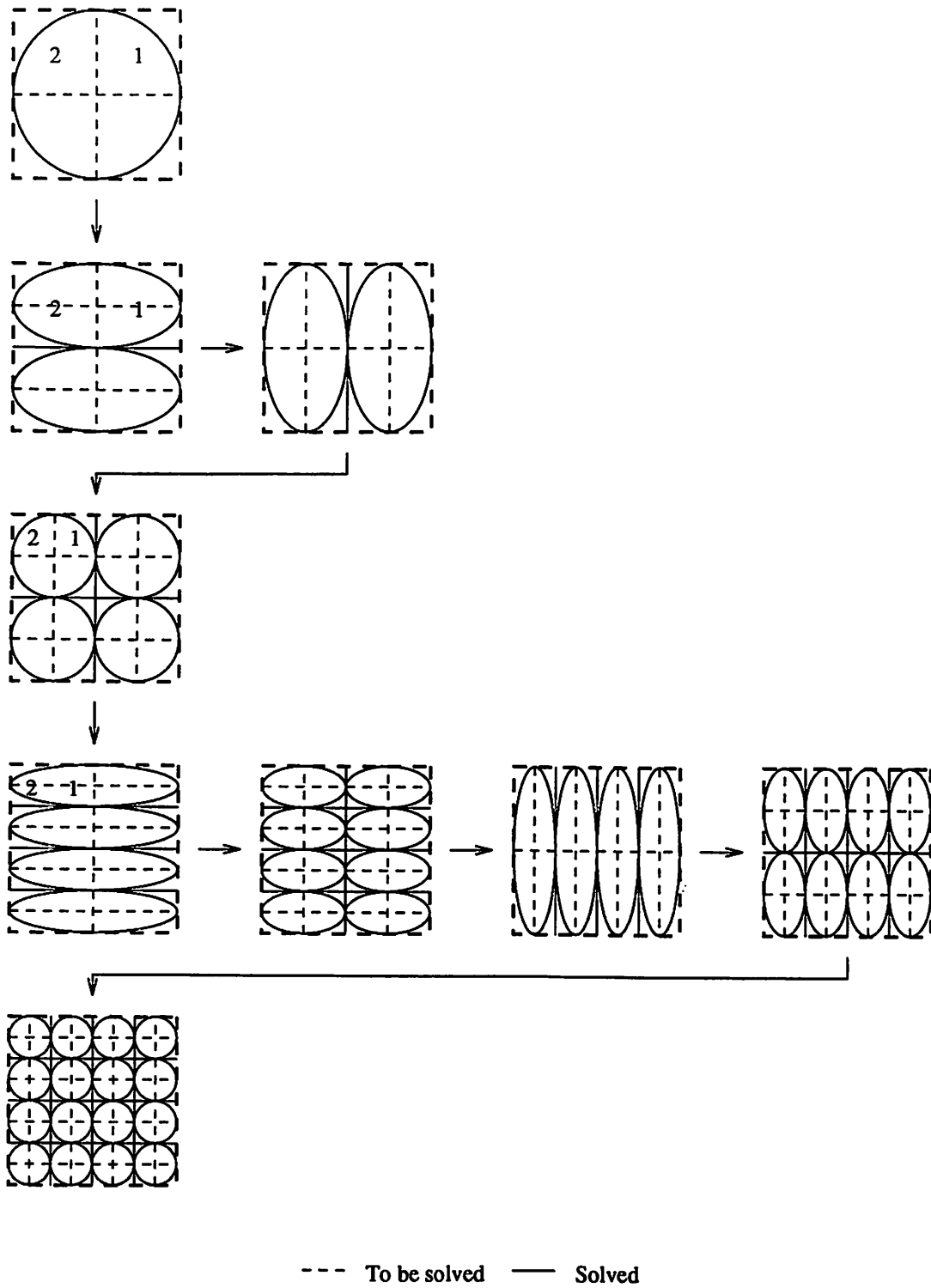


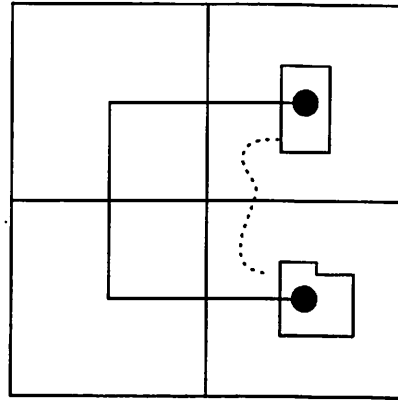
Figure 6.1: Merged approach subproblem abstraction sequence

set of pseudo-pin assignment abstractions and the next level of regular subproblem abstractions. Fortunately, a careful ordering of the pseudo-pin assignment subproblems provides this refinement without any extra effort.

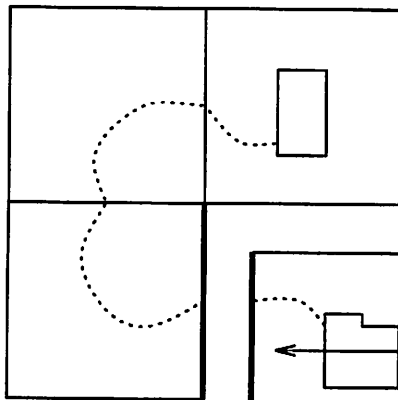
Feedback between placement and routing also results from using pseudo-pin locations to implement the terminal propagation constraints. Normally, external connection information is represented by dummy cells whose legal partition assignments are constrained heuristically based on the locations of the corresponding external connections. In this case, each pseudo-pin is treated as a cell and the constraints on admissible partition assignments reflect the range of possible pseudo-pin locations. Figure 6.2 illustrates how pseudo-pin positions bias placement after route assignment and pseudo-pin assignment. After route assignment, cells are biased towards the boundaries of pseudo-pins that are connected to the same nets and after pseudo-pin assignment, the placement bias is directed toward a specific portion of these boundaries.

Results for this merged approach have been compared with results for the separate phase approach. The examples used are the minimum area grid arrays of Chapter 3. The merged results were obtained as described above and the separate results were generated by applying the separate placement phase algorithm followed by the separate routing phase algorithm using the immediate pseudo-pin assignment method. No re-routing was performed during the execution of the separate routing phase procedure. Unfortunately, the performance of the merged approach is worse for all examples. Any benefit that may be derived from the pseudo-pin terminal propagation and the pseudo-pin assignment re-placement and re-routing is not enough to overcome the poor routing decisions made based on approximate placement data. Also, a limitation in the pseudo-pin assignment re-routing may also be a contributing factor.

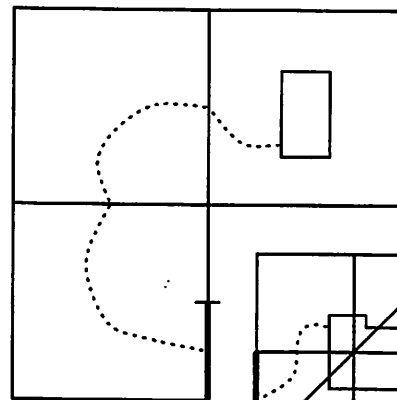
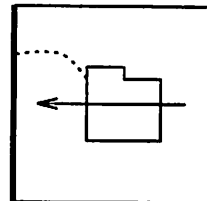
The suboptimality of route assignment decisions during the merged approach is verified by applying the same routing procedure used for the separate phase results to the placement generated by the merged method and comparing results. These results as well as the original comparisons are presented in Table 6.1. The separate phase algorithm results are listed under **Separate Phases**, the merged phase algorithm solutions are shown under **Merged Phases**, and the results generated by applying the separate phase routing procedure to the merged phase placements are under the heading **Second Route**. The basic comparison metric is the number of capacity violations in the final global routing graph. When no violations exist, total net length as measured by the semiperimeter of



(a)



(b)



(c)

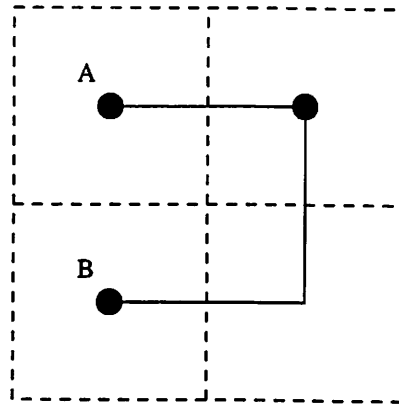
Figure 6.2: (a) parent problem solution (b) placement bias from routing solution (c) placement bias from pseudo-pin assignment solution

Example	Edge Capacity Violations			Net Length (λ)	
	Separate Phases	Merged Phases	Second Route	Separate Phases	Second Route
C880	5	31	7	—	—
C1355	9	88	11	—	—
misex3c	0	7	0	163422	174692
duke2	0	12	0	186062	200726
C1908	6	39	6	—	—
misex3	0	12	0	195043	197776
C3540	46	584	189	—	—
C5315	1106	1379	1142	—	—
C7552	589	1289	766	—	—
C6288	13	396	29	—	—

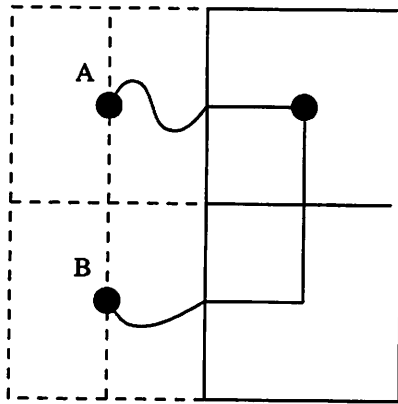
Table 6.1: Merged placement and routing versus separate phases

net bounding boxes is used. The merged approach produced significantly worse results than the separate phase approach. While applying the separate phase routing procedure to the merged placement generated much better results, the results are still worse than those obtained from separate placement and routing phases. The improvement confirms that poor route assignment decisions were made during the merged method. That the separate phase placement and the merged placement are different shows that the pseudo-pin terminal propagation and the pseudo-pin assignment rip-up, re-place, and re-route had an effect. Unfortunately, since the influence these factors have on the placement are based on the poor route assignment decisions, the effectiveness of these mechanisms for improving placement quality can not be determined.

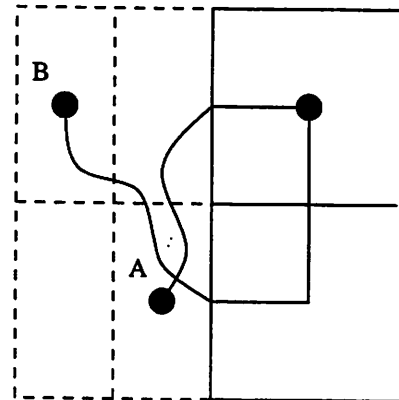
A limitation of the route assignment implementation may reduce the effectiveness of using pseudo-pin assignment abstractions to perform rip-up, re-placement, and re-routing. The intersection of a problem abstraction region with a net may result in multiple connected components of the net. The route assignment algorithms treat each connected component as a separate net. Normally this is fine because the nodes of each connected component were selected optimally by previous route assignments. However, after rip-up and re-placement, the optimal organization of a net's terminals into connected components may change and using the original structure may cause difficulties. An example of this situation is shown in Figure 6.3. Figure 6.3 (a) shows a route assignment for a particular



(a)



(b)



(c)

Figure 6.3: (a) route assignment on a problem abstraction (b) pseudo-pin assignment abstraction and connected components (c) suboptimal connected components after replacement

net on some problem abstraction. In a subsequent pseudo-pin assignment abstraction, this net will be treated as two nets corresponding to the two connected components shown in Figure 6.3 (b). Terminal A belongs to the upper connected component and terminal B belongs to the lower connected component. Unfortunately, after re-placement and despite the pseudo-pin terminal propagation bias, the cells containing terminals A and B may have moved such that current connected component structure is no longer optimal. This situation is shown in Figure 6.3 (c). Using the original connected component structure results in a subproblem that is more congested and more difficult. Ideally, the route assignment process on the pseudo-pin assignment abstraction would perform an analysis to re-group the connected components if necessary. However, even if this improvement were added, the route assignment decisions are still being made based on coarse placement information and the pseudo-pin assignment rip-up, re-placement and re-routing may not be able to remedy previous poor decisions.

6.3 Coupled Approach

The decomposition hierarchy can also be used to implement a coupled approach to placement and routing. The motivation for this variation is to use routing to influence subsequent placement and to improve the quality of the route assignment information by always basing the decisions on detailed placement information. The basic method is to perform a complete initial placement and then to perform re-placement after every level of routing refinement. The separate phase algorithms are used in each step and the routing feedback to placement occurs through the use of pseudo-pin terminal propagation.

Figures 6.4 and 6.5 show the sequence of subproblem abstractions solved for the coupled method. Figure 6.4 shows the initial detailed placement and the first level of routing and pseudo-pin assignment. The immediate method of pseudo-pin assignment is used to refine pseudo-pin locations. After each level of routing and pseudo-pin refinement, each subtree rooted at a current level routing subproblem is re-placed. This re-placement process allows routing feedback to placement. The terminal propagation for each subtree is relative to the nets defined in its corresponding root routing subproblem. The pseudo-pins present in these nets represent previous route assignment decisions and feed back routing information to placement through the propagation of their positions into subsequent placement abstractions. Figure 6.5 depicts the remaining subproblem sequence for the

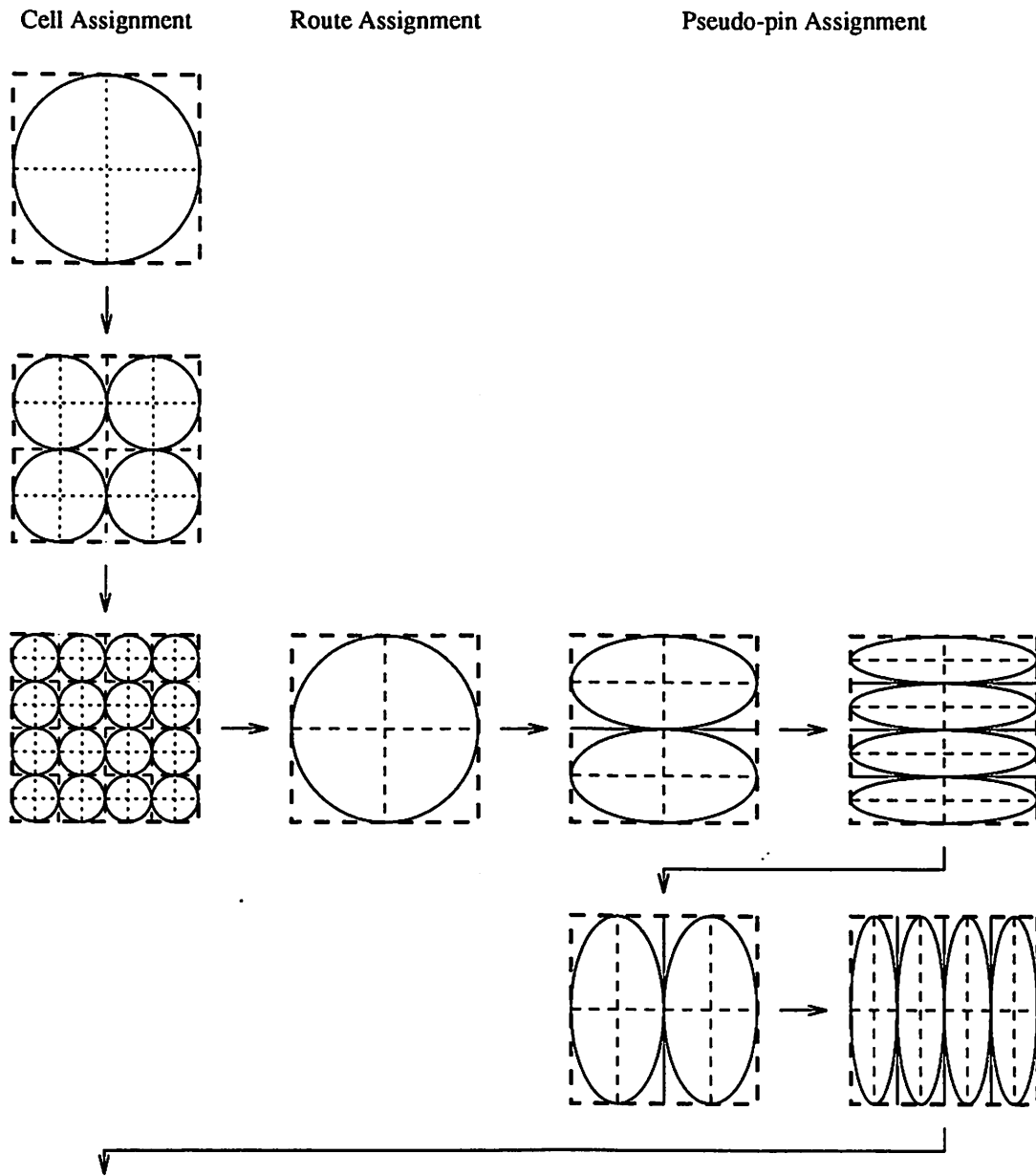


Figure 6.4: Initial coupled approach subproblem abstraction sequence

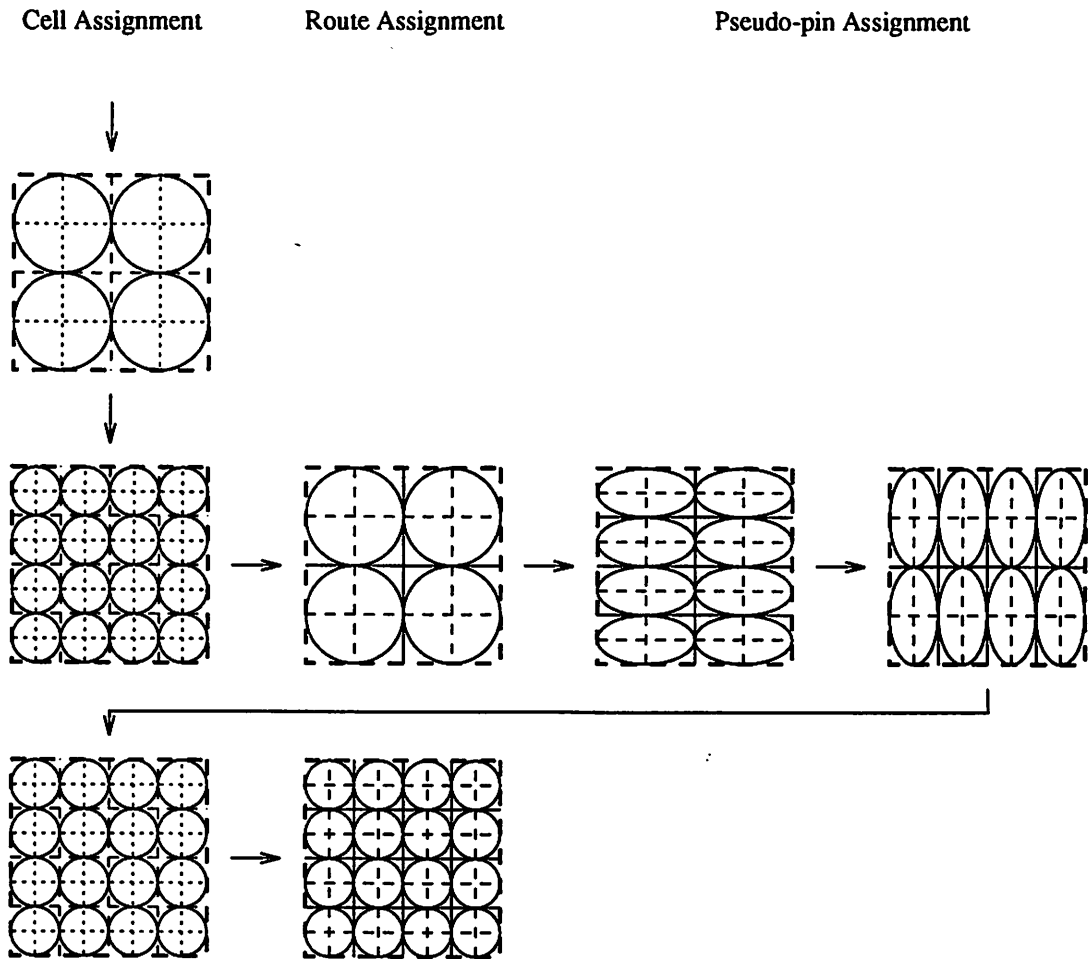


Figure 6.5: Remaining coupled approach subproblem abstraction sequence

Example	Edge Capacity Violations			Net Length (λ)		
	Separate Phases	Coupled Phases	Second Route	Separate Phases	Coupled Phases	Second Route
C880	5	7	6	—	—	—
C1355	9	23	6	—	—	—
misex3c	0	0	0	163422	167916	167197
duke2	0	4	0	186062	—	188065
C1908	6	16	8	—	—	—
misex3	0	0	0	195043	197943	197297
C3540	46	78	13	—	—	—
C5315	1106	1161	1035	—	—	—
C7552	589	723	571	—	—	—
C6288	13	73	9	—	—	—

Table 6.2: Coupled placement and routing versus separate phases

coupled approach and illustrates the re-placement between levels of routing refinement.

Results for the coupled approach are presented in Table 6.2. The examples are the minimum area grid arrays from Chapter 3. The metrics are edge capacity violations in the final global routing graph and semiperimeter bounding box net length as appropriate. The results obtained from the separate phases algorithms are based on the basic separate placement phase and a separate routing phase using the immediate method of pseudo-pin assignment and no re-routing. The coupled phases results were generated as described above and the second route category lists the results of applying the separate routing phase procedure to the placement generated by the coupled approach. Though the overall coupled approach produced worse results than the separate phase algorithms, it was able to produce better placements on several examples.

The poor performance of the couple approach compared to the separate phases approach is another manifestation of route assignment dependence on placement information. Poor route assignment decisions during the coupled method are confirmed by the universal improvement in results obtained by routing the placement generated by the coupled approach with the separate phase routing procedure. The coupled approach avoids the problem of using approximate placement information in the routing process by always having detailed placement data available. Unfortunately, the accuracy of these data can not be guaranteed. Actually, the opposite can be almost guaranteed. Since placement mod-

ifications based on routing decisions are desired, early route assignment decisions will be often based on placement configurations different from the final detailed placement.

Routing feedback to placement is possible and effective. In particular, the use of pseudo-pin terminal propagation to transmit routing information resulted in better placements on several examples. These examples are **C1355**, **C3540**, **C5315**, **C7552**, and **C6288**. The improvement in placement quality is measured relative to the separate routing phase algorithms and is confirmed by the improved results obtained by the application of the separate routing phase procedure to the coupled approach placements.

6.4 Hybrid Approach

A hybrid approach to combining placement and routing can be used to account for routing dependencies on detailed placement while still providing routing feedback to the cell assignment process. The method exploits the differences in placement requirements for the two portions of the route assignment procedure. The net group assignment portion only requires placement information refined to a level commensurate with the current routing grid abstraction, while the individual net assignment portion is highly dependent on final cell locations. The sequence of subproblem abstractions for the hybrid technique looks much like the sequence for the method of separate placement and routing phases. However, after each placement abstraction is solved, its corresponding routing abstraction is solved and congestion information from the net group assignment results are fed back to modify the placement. This process provides an opportunity to investigate more explicit feedback mechanisms based on modifying cell assignment algorithm parameters and constraints.

The sequence of subproblem abstractions solved for the hybrid approach is illustrated in Figures 6.6 and 6.7. First, cells are placed in a manner similar to the separate placement phase procedure. Terminal propagation is based on the original global nets and the solution of placement abstractions is iterated within levels. The difference is that solving each placement subproblem consists of iteration between the cell assignment and route assignment algorithms. The first application of the cell assignment algorithm uses the same parameters as the separate placement phase version. Subsequent applications use the results of the previous application as the initial partition and modified parameters based on the net group assignment solution of the corresponding routing subproblem. Iteration continues until the calculated cost stops improving. The cost metric used is the congestion

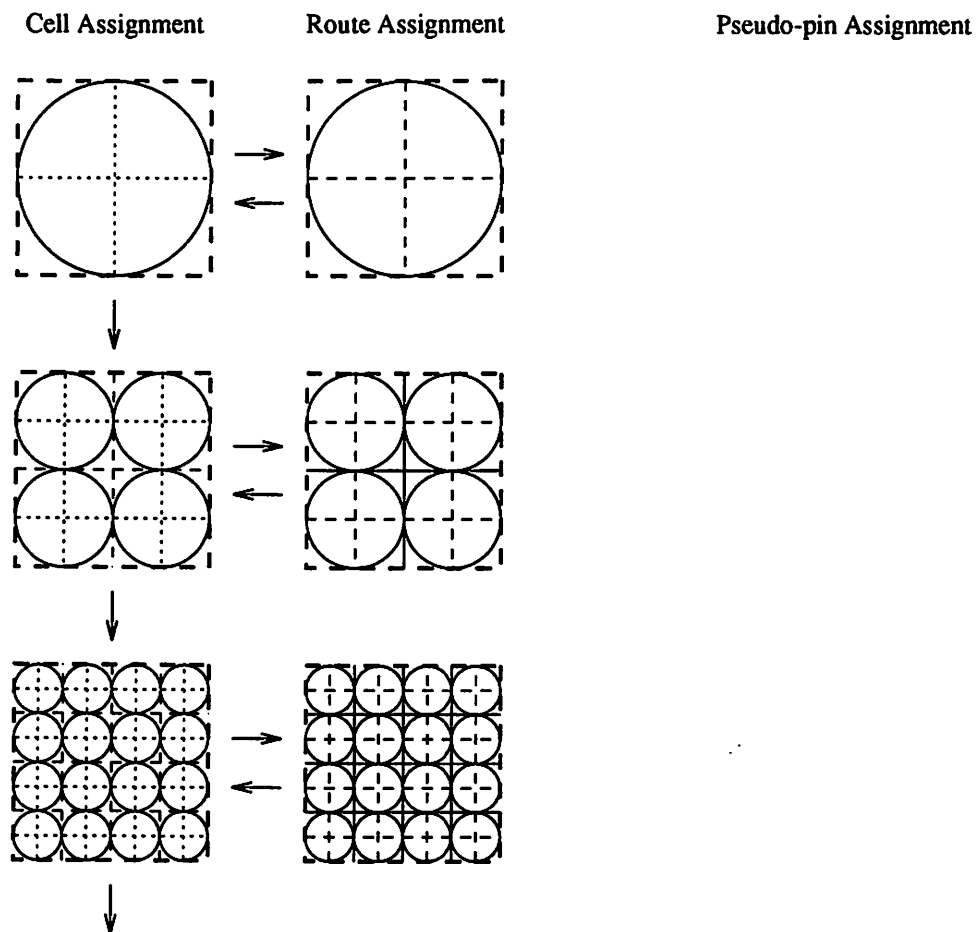


Figure 6.6: Initial hybrid approach subproblem abstraction sequence

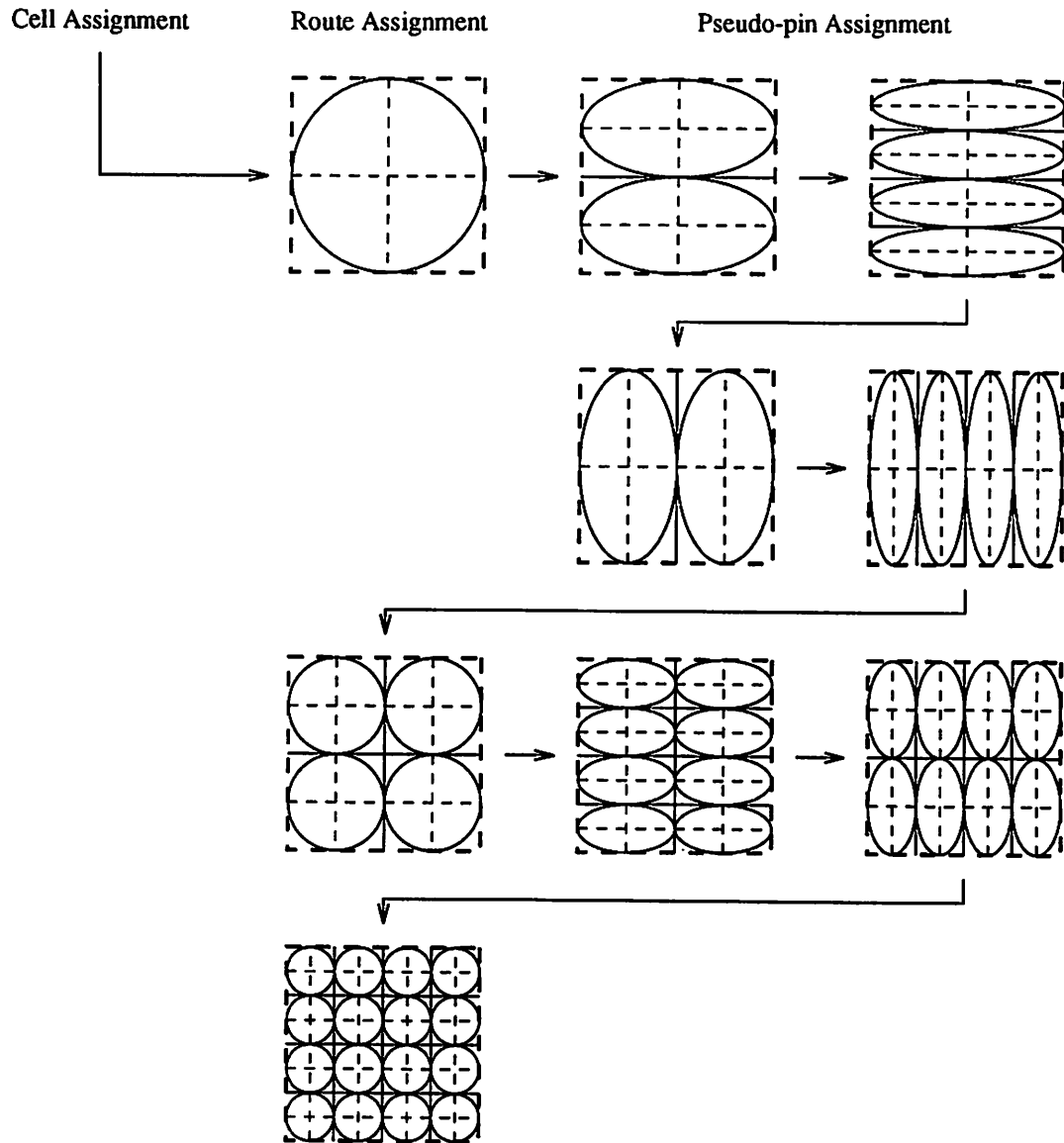


Figure 6.7: Remaining hybrid approach subproblem abstraction sequence

value of the most congested subproblem vertex plus the number of edge capacity violations in the current subproblem abstraction. Congestion estimation is discussed later. This portion of the hybrid approach is illustrated in Figure 6.6. The modification of cell assignment parameters represents a feedback path from the routing process. However, the feedback information is based only on routing predictions since the net group assignments are not kept for later use. Once the final detailed placement is known, the final route assignment decisions are made. This process is shown in Figure 6.7 and corresponds to the separate routing phase procedure using the immediate method of pseudo-pin assignment and no re-routing. The mechanism for feeding back routing information to the placement process is to modify cell assignment algorithm parameters and constraints. The parameters are the edge and vertex weights in the placement cost function and the constraints are the partition block target sizes. The parameters and constraints are modified heuristically based on usage and capacity estimates.

Edge weights represent the relative cost of using different edges of a problem abstraction. The cost of using an edge is set inversely proportional to the relative utilization of the edge:

$$weight(e_k) = 100 \frac{congestion(e_k)}{\sum_{e_i \in \mathcal{E}} congestion(e_i)}$$

where

$$congestion(e_k) = \frac{usage(e_k) + 1}{capacity(e_k)}.$$

The capacity of an edge e_k is calculated as in Section 5.1.1 and the usage of an edge is determined from the route assignment solution.

Vertex weights represent the relative cost of assigning degenerate nets to the different vertices of a problem abstraction. The cost for each vertex is proportional to the relative utilization or congestion of its corresponding subproblem:

$$weight(v_j) = 100 \frac{congestion(v_j)}{\sum_{v_i \in \mathcal{V}} congestion(v_i)}$$

where

$$congestion(v_j) = \frac{usage(v_j) + 1}{capacity(v_j)}.$$

Resource usage, $usage(v_j)$, in a subproblem vertex, v_j , is the sum of usage costs over all the subnets in the subproblem vertex. The usage cost of a subnet is estimated as one less than the number of subnet terminals. Each cell on a subnet is counted as one terminal and

all dummy terminal propagation cells on a subnet are counted collectively as one terminal. Subnets for each subproblem vertex are defined by the route assignment solution and each edge of a route configuration contributes one terminal to the appropriate subnet in each incident subproblem vertex. The capacity, $capacity(v_j)$, of a subproblem vertex, v_j , is defined as the sum of the feed-through and track capacity estimates for the corresponding super-grid location as described in Section 5.1.1.

The constants C_1 and C_2 define the relative importance of vertex and edge weights to the partition cut weight. Currently, C_1 equals one and C_2 is adjusted before each iteration to ensure that the minimum edge cost is always greater than the maximum vertex cost.

The other mechanism for changing placement decisions is the modification of partition block area targets based on congestion information. Congested subproblems are assigned fewer cells by setting

$$target(\mathcal{M}_{v_j}) = \sum_{k=1}^M weight(m_k) \times \frac{\frac{1}{congestion(v_j)}}{\sum_{v_i \in \mathcal{V}} \frac{1}{congestion(v_i)}}.$$

Congestion values are calculated as described earlier. The maximum error limit, ϵ , may be set to zero for better feedback control.

Comparisons between the hybrid approach and the use of separate phases is based on the minimum area array examples of Chapter 3. The performance metrics are number of edge capacity violations in the final global routing graph and total semiperimeter bounding box net length as appropriate. The separate phase approach uses the algorithms described in Chapter 4 as before. The separate phase routing procedure uses the immediate method of pseudo-pin assignment and does not employ any re-routing. The comparisons are presented in Table 6.3. The performance of the hybrid approach was measured for feedback through edge and vertex weight modification only (**Weight**), feedback through partition block target size modification only (**Area**), and feedback through both mechanisms simultaneously (**Both**). Re-routing was not performed for any of the hybrid approaches.

Using the proposed feedback mechanisms produced comparable or better solutions on almost all of the examples. The area-only feedback option generated better results for all the examples and the weight-only feedback option and the combination of feedback methods produced comparable or better results on all the examples except for example **C7552**. When the methods were able to find feasible solutions, the total net length values achieved were within 3% to 4% of each other.

Example	Edge Capacity Violations				Net Length (λ)			
	Separate	Weight	Area	Both	Separate	Weight	Area	Both
C880	5	1	2	5	—	—	—	—
C1355	9	3	7	3	—	—	—	—
misex3c	0	0	0	0	163422	168836	163099	167928
duke2	0	0	0	0	186062	192034	185537	192674
C1908	6	4	3	4	—	—	—	—
misex3	0	0	0	0	195043	203497	194959	202432
C3540	46	3	21	3	—	—	—	—
C5315	1106	769	954	715	—	—	—	—
C7552	589	638	566	643	—	—	—	—
C6288	13	4	8	6	—	—	—	—

Table 6.3: Hybrid placement and routing versus separate phases

The feedback mechanisms presented have potential for improving layout solutions, but more investigation is required to understand their interaction and relation to algorithm performance. Two areas for further examination are the relative importance of edge versus vertex weights and control of the cell and route assignment iterations.

The relative importance of edge and vertex weights in the placement cost function is controlled by the values of C_1 and C_2 . When all the vertex weights are zero, the placement cost function calculates a weighted partition cut cost. Historically, minimizing cut cost has correlated well to minimizing net length and minimizing net length has been a useful heuristic goal for producing good placements. Vertex weights correspond to congestion within partitions and the goal is to distribute this internal congestion evenly without adversely affecting the external congestion as represented by the partition cut cost. Implementing this heuristic requires that the product of the maximum vertex weight and C_1 is less than the product of the minimum edge weight and C_2 for a given application of the cell assignment algorithm. While C_1 and C_2 are set dynamically to insure this condition, doing so makes controlling the cell and route assignment iterations more difficult. Furthermore, given fixed partition capacities, it may be possible that the desired partition will have a partition cut cost that is higher than the minimum possible in order to alleviate congestion in a particular partition block. This means that some increase in external congestion could be allowed.

Another issue is the control of the iterative improvement both within a subproblem abstraction and within a level of the hierarchy. Currently, the control is based on minimizing the maximum congestion of subsequent subproblem abstractions. The goal of this heuristic is to avoid creating highly localized congestion. Unfortunately, attempting this optimization using a simple, greedy iterative process may not be the most effective. Also, higher congestion in a particular subproblem is not undesirable unless there is insufficient capacity to handle the congestion. The current heuristic attempts to check this condition, but its effectiveness is limited by the accuracy of the capacity estimates. Further research is required to obtain better capacity and congestion predictors for feedback iteration control.

Chapter 7

Conclusion

A layout system that uses hierarchical decomposition based on a 2×2 grid-graph data model has been implemented to explore ways of combining or merging placement and routing to produce better layout solutions. The major contributions of these experiments are a better characterization of the relationship between routing and placement information, confirmation of the usefulness of a common data model for improving communication and feedback between layout phases, identification of effective placement and routing feedback mechanisms, and an improved routing algorithm.

An understanding of the dependencies between placement and routing is crucial to any attempt at optimizing the combined placement and routing problem, and the structure induced by the hierarchical decomposition has proved useful for analyzing the dependence of routing decisions on placement information. Based on the experimental results, good route assignments for individual nets require final detailed placement information. Though the simple statement of this dependency may be intuitive, the more important contribution is the characterization of how quickly routing performance degrades with decreased placement information accuracy. Further examination of the routing procedure led to the realization that the dependence was not the same for all routing decisions. In particular, the assignment of routes to individual nets is highly dependent on detailed placement information while the assignment of routes to net groups is not. This difference can be exploited to feed back useful routing information to the placement process.

The usefulness of a common data model is demonstrated by the ease of formulating feedback mechanisms and the different approaches to combining placement and routing. The convenient calculations of routing usage and capacity correspond directly to placement

partition block boundaries or areas. In particular, the feedback modification of placement cost function weights is a good example of the effectiveness of the common data model. Using a common data model also results in the same problem hierarchy for both placement and routing. None of the experiments in combining placement and routing could have been implemented as easily without this feature.

A contribution to improving the standard layout paradigm is the identification of effective feedback mechanisms from routing to placement. These mechanisms use routing decisions to influence terminal propagation and to modify placement algorithm parameters. Route assignments induce pseudo-pins in the problem hierarchy that can be used instead of dummy cells to represent connections to cells outside of the current placement abstraction. Since the legal placement partitions of the pseudo-pins are determined by the assigned routes, subsequent cell placement is based on the proposed routing rather than simpler relative position heuristics. Experimental results show that it is possible to generate better placements using this technique. However, the method is dependent on the quality of the proposed individual net route assignments and making these route assignment decisions early in the placement process is difficult because final placement locations are not known accurately. Routing decisions can also be used to modify edge and vertex weights in the cell assignment cost function and to change the cell assignment partition block target size constraints. These feedback paths are more useful because they only require the routing decisions on assignments to groups of net types. These decisions require a minimal amount of placement information and are not dependent on final placement positions. These feedback methods were also able to produce better results than the standard layout paradigm of separate placement and routing phases, though more research is required to understand the trade-offs and relationships between the different placement parameters and routing congestion.

A side benefit of these experiments on merging placement and routing has been the development of an improved hierarchical router. This router retains the basic speed and net-ordering independence advantages of previous hierarchical approaches while also achieving large reductions in the numbers of routing capacity violations when routing fixed areas. The improvement is obtained by solving a linearly constrained nonlinear integer program that distributes wiring patterns in proportion to the available routing resources. The efficient solution of this integer program is possible because standard solution methods are able to exploit the structure of the corresponding linearly constrained nonlinear program

to find naturally solutions that are integer or close to integer.

The analysis of the relationship between placement and routing and the subsequent experiments on various methods of combining and communicating feedback information between the two phases have indicated potential techniques for ameliorating the effects of the artificial decomposition of the layout problem into separate phases. Continued research in this area should produce further progress towards more global optimization of the entire layout problem.

Bibliography

- [1] Glenn Adams. *Non-Sequential Tool Interaction in the Context of Sea-of-Gates Module Generation*. PhD thesis, University of California, Berkeley, to be published.
- [2] Alfred E. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [3] H. J. Prömel B. Korte and A. Steger. Combining partitioning and global routing in sea-of-cells design. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 98–101, 1989.
- [4] Shekar Bapat. *A Sharp Methodology for VLSI Layout*. PhD thesis, University of Virginia, 1993.
- [5] M. Breuer. *Design Automation of Digital Systems*, volume 1. Prentice-Hall, Englewood Cliffs, NJ, 1972.
- [6] Randall J. Brouwer and Prithviraj Banerjee. PHIGURE: A parallel hierarchical global router. In *Proceedings of the 27th Design Automation Conference*, pages 650–653, 1990.
- [7] Michael Burstein. An approach to design automation of custom LSI chip layout based on heuristic planarization and annular imbedding. In *Proceedings of IEEE International Conference on Circuits and Computers*, volume 1, pages 1056–1059, October 1980.
- [8] Michael Burstein. A non-‘placement/routing’ approach to automation of VLSI layout design. In *Proceedings of IEEE International Symposium on Circuits and Systems*, volume 3, pages 756–759, May 1982.

- [9] Michael Burstein, Se June Hong, and Richard Pelavin. Hierarchical VLSI layout: Simultaneous placement and wiring of gate arrays. In *Proceedings of the International Conference on Very Large Scale Integration*, pages 45–60, August 1983.
- [10] Michael Burstein and Richard Pelavin. Hierarchical wire routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-2(4):223–234, October 1983.
- [11] Wayne A. Christopher. Mariner — A sea-of-gates layout system. Master's thesis, University of California, Berkeley, May 1989.
- [12] Wei-Ming Dai and Ernest S. Kuh. Simultaneous floor planning and global routing for hierarchical building-block layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(5):828–836, September 1987.
- [13] W.A. Dees and P.G. Karger. Automated rip-up and reroute techniques. In *Proceedings of the 19th Design Automation Conference*, pages 432–439, 1982.
- [14] A. E. Dunlop and B. W. Kernighan. A procedure for layout of standard cell VLSI circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(1):92–98, January 1985.
- [15] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, pages 175–181, 1982.
- [16] David S. Harrison, Peter Moore, Rick L. Spickelmier, and A. Richard Newton. Data management and graphics editing in the berkeley design environment. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 20–24, November 1986.
- [17] Mitsuru Igusa, Mark Beardslee, and Alberto Sangiovanni-Vincentelli. ORCA: A sea-of-gates place and route system. In *Proceedings of the 26th Design Automation Conference*, pages 122–127, June 1989.
- [18] D.S. Johnson. The NP-completeness column: An ongoing guide. *J. Algorithms*, 3:381–395, 1983.

- [19] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, February 1970.
- [20] Balakrishnan Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, C-33(5):438–446, May 1984.
- [21] Ulrich Lauther. A min-cut placement algorithm for general cell assemblies based on a graph representation. In *Proceedings of the 16th Design Automation Conference*, pages 1–10, 1979.
- [22] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.
- [23] Lorraine S. Layer. Analysis of sea-of-gates template and cell library design issues. Memorandum UCB/ERL M88/8, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA, 94720, January 1988.
- [24] Brian D. N. Lee. Experiments in hierarchical routing of general areas. Master's thesis, University of California, Berkeley, May 1989.
- [25] D. T. Lee, S. J. Hong, and C. K. Wong. Number of vias: A control parameter for global wiring of high-density chips. *IBM Journal of Research and Development*, 25(4):261–271, July 1981.
- [26] Malgorzata Marek-Sadowska. Global router for gate array. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, pages 332–337, October 1984.
- [27] Rolf Müller. A new approach for simultaneous floorplanning and global wiring. *Methods of Operation Research*, 62:287–289, 1990.
- [28] Bruce A. Murtagh and Michael A. Saunders. MINOS 5.1 user's guide. Technical Report 83-20R, Systems Optimization Laboratory, Stanford University, 1983.
- [29] Michiroh Ohmura, Shin'ichi Wakabayashi, Yoshihiro Toyohara, Jun'ichi Miyao, and Noriyoshi Yoshida. Hierarchical floorplanning and detailed global routing with routing-based partitioning. In *Proceedings of the International Symposium on Circuits and Systems*, volume 2, pages 1640–1643, May 1990.

- [30] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
- [31] Prabhakar Raghavan. Randomized rounding and discrete ham-sandwich theorems: Provably good algorithms for routing and packing problems. Technical Report UCB/CSD 87/312, Computer Science Division, College of Engineering, University of California, Berkeley, CA, 94720, July 1986.
- [32] C. Sechen et al. *TimberWolf 6.1 Manual*. Yale University, September 1991.
- [33] E.M. Sentovich, K.J. Singh, C. Moon, H. Savoj, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Sequential Circuit Design Using Synthesis and Optimization. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, October 1992.
- [34] K. Shookar and P. Mazumber. VLSI cell placement techniques. *ACM Computing Surveys*, 23(2):143–220, June 1991.
- [35] Hyunchul Shin and Alberto Sangiovanni-Vincentelli. A detailed router based on incremental routing modifications: Mighty. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(6):942–955, November 1987.
- [36] Eugene Shragowitz, Jong Lee, and Sartaj Sahni. Placer-router for 'sea of gates' design style. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, pages 330–335, October 1987.
- [37] Jiri Soukup. Circuit layout. *Proceedings of the IEEE*, 69(10):1281–1304, October 1981.
- [38] J. Soukup and J. C. Royle. On hierarchical routing. *Journal of Digital Systems*, 5(3):265–289, 1981.
- [39] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 57(3):91–101, June 1983.
- [40] P. R. Suaris and G. Kedem. Standard cell placement by quadrisection. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, pages 612–615, October 1987.

- [41] Peter R. Suaris and Gershon Kedem. A quadrisection-based combined place and route scheme for standard cells. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(3):234–244, March 1989.
- [42] Antoni A. Szepieniec. Integrated placement/routing in sliced layouts. In *Proceedings of the 23th Design Automation Conference*, pages 300–307, 1986.
- [43] Ping-San Tzeng. *Integrated Placement and Routing for VLSI Layout Synthesis and Optimization*. PhD thesis, University of California, Berkeley, 1992.
- [44] Ping-San Tzeng and Carlo H. Séquin. Codar: A congestion directed general area router. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 30–33, 1988.
- [45] Saeyang Yang. *Logic Synthesis and Optimization Benchmarks User Guide Version 3.0*. Microelectronics Center of North Carolina, January 1991.