EXPERIMENTS IN HIERARCHICAL ROUTING OF GENERAL AREAS

by

Brian D. N. Lee

Memorandum No. UCB/ERL M90/17

13 March 1990

EXPERIMENTS IN HIERARCHICAL ROUTING OF GENERAL AREAS

by

Brian D. N. Lee

Memorandum No. UCB/ERL M90/17

13 March 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

EXPERIMENTS IN HIERARCHICAL ROUTING OF GENERAL AREAS

by

Brian D. N. Lee

Memorandum No. UCB/ERL M90/17

13 March 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Acknowledgements

I would like to thank my research advisor, Prof. Richard Newton, for his initiation of this project and for his generous and constant support and patience throughout its duration. His vision and ideas kept me motivated and helped me learn to remain focused on the "bigger" picture.

This work was sponsored in part by the Hewlett-Packard Company, the LSI Logic Corporation, the Digital Equipment Corporation, the National Science Foundation (under contract #ECS 8430435), and the Defense Advanced Research Projects Agency (under contract #N00039-C-87-0182). Their support is gratefully acknowledged.

I would also like to thank the CAD group for providing such an open, creative, and stimulating research environment. In particular, I wish to thank Jeff Burns, Wayne Christopher, David Harrison, Tom Laidig, Peter Moore, Tom Quarles, Greg Sorkin, and Rick Spickelmier for all their time and help. My discussions with them have always been useful and I have learned and grown much from their advice and example.

I am also grateful to my family and friends for putting up with me and my long work hours. They knew when to let me "go kamikaze" and when to drag me from my cubicle. To them, I owe my sanity.

Finally, and most importantly, I would like to thank my parents for their constant love and faith.

Contents

List of Figures

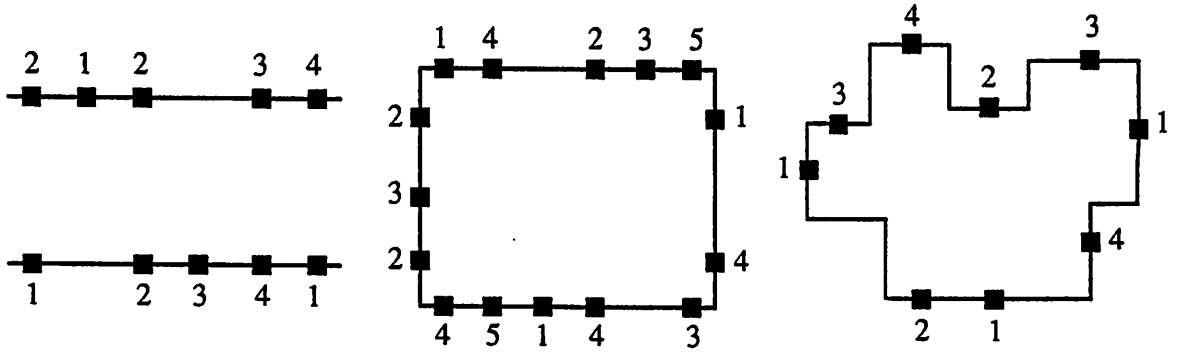
List of Tables

Chapter 1

Introduction

The complexity and size of integrated circuit designs has increased dramatically in recent years. Accordingly, there is a need for design automation tools that can handle ever larger and more complex routing problems. The problems of interest have evolved from simple channels to switchboxes to general areas. Currently, there is great interest in solving general area routing problems because of the rising popularity of new design methodologies and the emergence of new manufacturing technologies. For example, the Sea-of-Gates design style with many layers of high-quality interconnect requires over-the-cell routing to achieve high circuit densities. The structure of such routing problems on the channelless gate array is clearly in the domain of general area routers. Similarly, the next generation PC board technology using silicon-on-silicon modules will also require a general area router. Because these complex routing problems are a manifestation of an overall increase in design complexity, routers must not only handle these general area problems but also interact with other design tools in a more sophisticated manner. In particular, the feedback between routing programs and placement programs must become more explicit and any approach developed for general area problems must take this interaction into consideration.

In this report, hierarchical decomposition is examined as a way to attack the general area routing problem. This divide-and-conquer method is a useful technique for managing complexity and in recent years has been applied to various routing problems. In particular, there are examples of this concept in channel routers [BP82, BP83a, HM85, HM89], switchbox routers [BP83b], and global routers [MS84, LTW86, MS86, Lau87]. This work describes a set of experiments that outlines the extension of these applications to general area routing problems.



(a) channel (b) switchbox (c) rectilinear polygon

Figure 1.1: Historical progression of routing problems

This work originated as part of a large sea-of-gates project, called Mariner [Lay88, Chr89]. The ultimate goal of the Mariner project is to combine placement and routing in a common, hierarchical data structure. In this way, local modifications to both placement and routing (e.g. rip-up-and-re-route and rip-up-and-re-place) can be performed concurrently and iteratively.

Since the project began, the utility of the router and its hierarchical paradigm for other problems, such as the silicon-on-silicon packaging problem [HCBA82], has become clear. Future work with the router will also be directed to this problem.

1.1 Problem Definition

The routing problem is to find a feasible connection of all the pins of each net in the routing region such that some cost function is minimized. For more general routing problems (Figure 1.1(b) and (c) above), net length, number of vias, and critical signal delay are typical examples of desired cost function components.

Routing problems may be made harder by changing various constraints. Examples of these constraints include the specification of the routing region, the permissible pin positions, the number of routing layers, and the priority of a given net. Figure 1.1 shows a historical progression of routing problems with respect to the complexity of the routing region boundary. The pins to be connected in a given net are indicated with the same number. The complexity of each of these cases may also be affected by allowing routing obstacles,

by changing the number of available routing layers, by allowing pins within the interior of the routing region, by removing routing-grid restrictions in the problem specifications, etc.

Figure 1.1(a) shows a classic channel routing problem. It was one of the earliest and simplest instances of the routing problem to be investigated extensively [HS71, Deu76, RF82, YK82, RSVS85]. In this case, the routing region boundary is defined by two parallel line segments and pins are specified at positions along these segments. Nets may be assigned to enter or exit from the open sides of the channel, but neither the position nor the relative order of these net crossings are specified. The distance between the segments may be varied as necessary to accommodate the routing, but the objective is to minimize this distance and thus minimize the area of the channel.

Figure 1.1(b) shows a classic switchbox routing problem [HO84, MS85, Luk85, Joo85, SSV87]. In this case, the routing region is specified by a rectangle. It can be thought of as a channel routing problem where the distance between the sides of the channel is fixed and where the pins are specified at fixed positions along the open sides of the channel. This is a much harder problem than the channel routing problem because the router is not allowed to increase the amount of available routing area and because the pins are specified on all four sides of the region. Thus, the primary goal is to find a feasible solution within the specified area.

Both the channel routing problem and the switchbox routing problem may be made harder by allowing the routing region to take on more irregular shapes. For example, an irregular channel routing problem could be created by defining a routing region with roughly parallel rectilinear path segments [DAK85]. This problem instance would be classified somewhere in between the classic channel routing problem and the classic switchbox problem with respect to routing region boundary complexity. If a similar change is applied to the switchbox problem of Figure 1.1(b), then a routing region results that is specified by a rectilinear polygon as shown in Figure 1.1(c). This leads to the general area problem [DAK85, SSV87].

The general area routing problem is defined as an instance of the routing problem with the following constraints on feasible solutions. Each available routing location may be occupied by exactly one net and these legal routing areas are constrained as follows.

- The routing region is represented by different, arbitrary, rectilinear polygons on each of its multiple routing layers.

- The region may have any number of routing obstacles which are also defined by arbitrary, rectilinear polygons on a per layer basis.
- All connection paths specified on a particular layer must be contained within that layer's corresponding boundary polygon and must not intersect that layer's corresponding obstacle polygons.
- The definitions of *occupy*, *contain* and *intersect* are specified so as to satisfy design rule constraints.
- Pins may be positioned anywhere within the region or on its borders.
- The layers of the geometry representing the implementation of each pin specify the legal routing layers for that pin.

An example of a three-layer general area problem is shown in Figure 1.2. In this figure, the boundary polygon and the obstacle polygons for each layer have been combined into a single polygon with holes. The holes in each polygon define the routing obstacles on the corresponding layer. Also, though the pins are shown as solid abstractions, they actually have size and associated layers.

1.2 Applications and Motivation

1.2.1 Sea-of-Gates Design Style

The Sea-of-Gates design style [HWD+85, HPE+86, CYK+87, BM87] is an excellent example of the need for general area routers. In this methodology, the routing must connect cell metalization templates that have been placed on a channelless gate array. Since a primary goal of the design style is high transistor utilization, the cell templates are placed quite densely and the usual channel or switchbox routing areas do not exist. Thus, the router must make use of all available routing area, including regions that run over or through the cells. This presents the router with a problem containing an enormous number of arbitrary blockages and pins that are only specified at positions within the routing region. Also, the trend in recent systems is towards using technologies with three or more metal layers and the router must effectively handle this increase in the number of routing layers. This complex topology is a general area problem and its efficient solution is critical to the effectiveness of the design style.

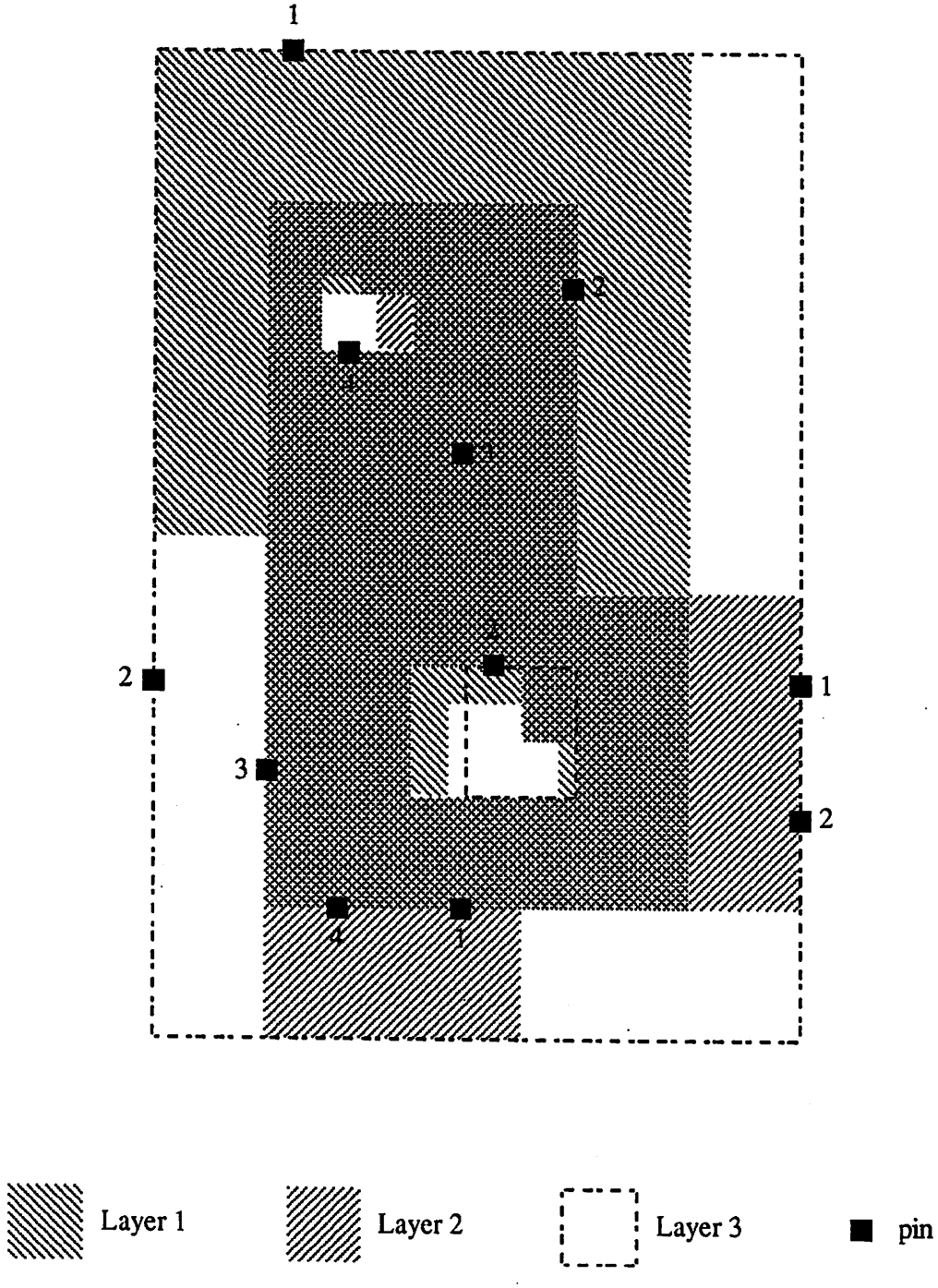


Figure 1.2: Example of a general area routing problem

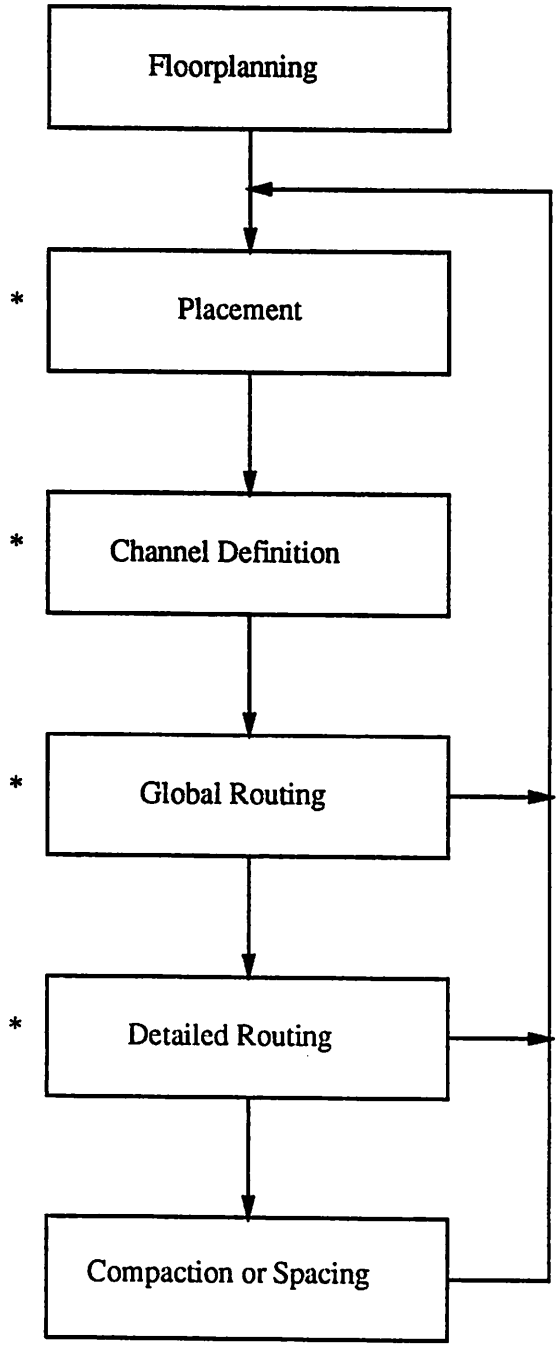
1.2.2 Silicon-on-Silicon Modules in PC Boards

The use of silicon chips embedded on silicon substrates which are in turn mounted on printed circuit boards is predicted for the next generation of board integration technology. Wiring between chips on the same chip-carrier substrate will be implemented using the same style of metallization processes as used on the individual chips (though the metal and insulator will be thicker to maintain good transmission line properties). The individual chips define arbitrary routing obstacles (on one or more layers, depending on details of the particular implementation strategy) and internal pin locations, resulting in a area routing problem with characteristics similar to the Sea-of-Gates situation. Thus, the problem of determining the routing for the interchip connections in silicon-on-silicon modules is also a general area routing problem.

1.2.3 Routing Unification and Routing Feedback to Placement

Current layout systems break down the layout problem into a series of sequential phases as shown in Figure 1.3. Each phase addresses a more detailed view of the problem based on the global decisions of the previous phase. Unfortunately, layout problems are global optimization problems and in an ideal system, information discovered in the more detailed phases should be fed back to the more global phases. While this may be simulated by performing manual iterations of the phases, further advances in the quality of layout solutions will require more automatic interaction and feedback between the different phases or equivalently, the different design tools. One goal of the Mariner project is to experiment with integrated representation and operation of all of the phases of layout marked in Figure 1.3[1]. In the context of routing, this means that a unified approach to both global and detail routing should be taken and that the routing process must interact with the placement process. The hierarchical decomposition approach to routing provides a way to integrate both global and detail routing. Each succeeding level of the decomposition hierarchy represents routing that is less global and more detailed in nature. Also, the hierarchy provides a powerful structure in which to pass information between the different levels of routing. Furthermore, the decomposition approach also provides useful information for router and placer communication. As described later in this report, the geometric decomposition based on topological and density metrics rapidly identifies congested or critical areas that need

[1]Channel definition becomes routing area specification in the general problem



* Mariner attempts to integrate these steps into a single hierarchical and iterative set of operations [Chr89]. A method of integrating global and detailed routing is proposed by [TS88].

Figure 1.3: A typical layout-system design flow

adjustment by the placer. Also, by controlling the extent of the hierarchical decomposition, i.e., limiting the depth of the hierarchy, information can be provided to the placement program at an appropriate level of abstraction with a the minimal amount of effort. In the model proposed for the Mariner project, the placer will execute the router as a subprogram on specific regions of the chip (defined by the hierarchical decomposition) and thus will be able to specify a decomposition limit commensurate with the current granularity of the placement.

1.2.4 Future Methodologies

The development of new design methodologies and new design tools is driven by the need to handle larger and more complex designs. To squeeze more functionality onto a chip, less area must be dedicated to routing in the form of channels and switchboxes and more area must be reclaimed from the unused areas in, around, and over the functional blocks or transistors. A general area router will be needed to achieve this goal and a hierarchical approach to the problem will be required to deal with the size complexity of the next generations of designs. Thus, an understanding of hierarchical routing of general areas will be valuable in adapting to future design methods and problems. In addition, a hierarchical approach of the form described here should lend itself naturally to a multicomputer implementation for problems of high complexity.

1.3 Previous Work

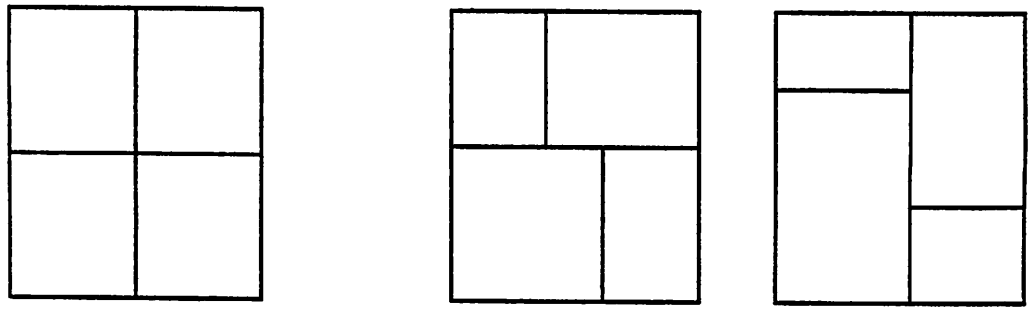
The work in formally applying hierarchical decomposition to routing was published by Burstein and Pelavin [BP82, BP83a, BP83b]. Their basic approach reduces the routing problem to the problem of routing on a $2 \times n$ grid [BP82, BP83a] or a 2×2 grid [BP83b] and then consistently applies a divide-and-conquer strategy. The $2 \times n$ approach is targeted for channel routing problems. Since little is gained from dividing a channel into two parallel subchannels of the same height, this is the most natural formulation of the problem. In this method, nets are routed one at a time on the $2 \times n$ grid, using a dynamic programming algorithm to find a minimal cost Steiner tree for each net. In the 2×2 approach, nets are classified by the number of grid cells which contain pins of the net. Since a 2×2 grid has four grid cells, the number of major net classifications is three, corresponding to nets with 2, 3, or 4 pins in different grid cells. The possible routes that these types of nets may

take on a 2×2 grid are enumerated and at each stage in the divide-and-conquer process, nets are assigned to specific patterns. Finding the number of nets that follow particular pattern on the 2×2 grid is formulated as an integer programming problem. A standard linear programming solution is invoked, the results are rounded, and then the assignment of a specific net of a given type to a particular pattern is completed heuristically. When the 2×2 grid abstraction becomes equal to the actual routing grid, the solution to the current subproblem represents the corresponding portion of the final, total solution. An advantage of this method is that the size of the integer programming problem is fixed and is independent of the number of nets. The dynamic programming method may be used instead of the linear programming method or it may be used in conjunction with the linear programming method to resolve overflows. Both the integer programming formulation and the dynamic programming method depend on the assumption of the standard two-layer wiring model. The standard two-layer wiring model requires that horizontal and vertical wire segments be assigned to different routing layers.

Marek-Sadowska [MS84] described a global router that uses a bottom-up hierarchical approach rather than a top-down approach to avoid the problems of passing bad decisions down the hierarchy. A re-router is applied as a second phase after the bottom-up hierarchical router to resolve overflows.

Hachtel and Morrison [HM85, HM89] provided an extension to $l \times m \times n$ grid graphs where l is the number of layers, m is the number of rows, and n is the number of columns. They use a $l \times 2 \times n$ decomposition and a combination of constructive placement and dynamic programming to perform the routing. First, a fast initial constructive placement of nets is performed on the $l \times 2 \times n$ abstraction. Any remaining unrouted nets are processed one net at a time using a dynamic programming algorithm to solve the Steiner Problem on the $l \times 2 \times n$ grid. Their dynamic programming formulation allows any specifiable set of wiring rules and thus removes the necessity for assuming the standard wiring model.

Luk, Tang, and Wong [LTW86] presented a global router for macro-cell layout that extends the 2×2 grid integer programming problem of [BP83b] to a generalized four-partition. The difference between the two partition models is shown in Figure 1.4. The 2×2 grid of Figure 1.4(a) corresponds to performing a quadrisection while the decompositions shown in Figure 1.4(b) are examples of possible four-partitions. That is, the four-partition divides a region into four portions without specifying any relationship between the sizes of each of the portions. Their router requires that the floorplan be a slicing structure [Ott82],



(a) quadrisection (2x2 grid) (b) four-partitions

Figure 1.4: Quadrisection versus four-partition

since the decomposition hierarchy corresponds directly to a balanced slicing tree of the design.

More recent global routing variations by Lauther [Lau87] and Marek-Sadowska [MS86] use a two-partition model rather than a four-partition model. In these methods, the routing at higher levels in the hierarchy is performed by assigning pseudo-pins to the partition boundary. Typically, this assignment problem is solved as a classical linear assignment problem.

In all of the above work, the routing region is decomposed first. This region decomposition then either explicitly or implicitly causes a decomposition of the nets into subnets corresponding to the newly created subregions. In some other variations, [PSK85] and [WM86], only the nets are decomposed hierarchically.

1.4 Current Research

In this work, the application of hierarchical decomposition concepts to general routing areas is investigated. In particular, an attempt is made to preserve and improve the top-down flow of information represented by the gradual refinement of the problem abstractions. Hierarchical decomposition involves partitioning the original problem into subproblems and then solving each of the subproblems in an optimal order. The goal then is to determine how best to partition the general area routing problem into subproblems and how to solve the resultant subproblems. The hierarchical decomposition process results

in a tree of subproblems or a routing hierarchy. A rip-up and re-route method must be supported that works at any level within this routing hierarchy since it is unlikely that any constructive approach will be 100% successful all the time.

The optimal physical partitioning of routing regions is the primary thrust of the work described in this report. The concept of an "intelligent" cut path that partitions the routing region into subproblems which are optimal with respect to reducing problem complexity and increasing the ease of routing in subsequent steps is investigated. This differs from most of the previous work in that an arbitrary rectilinear path is used to divide a region instead of a straight line and in that "intelligent" topology-based cut path selection heuristics are used instead of simple bisection- or quadrisection-oriented selection heuristics. To round out these experiments on partitioning general area routing problems, the subsequent net partitioning and pin assignment phases are also examined, though additional work remains in these areas.

In the previous work, hierarchical decomposition was used for either global routing or detailed routing, but not both at the same time. Here, a divide-and-conquer strategy is applied that implements global routing, detail routing, and the various levels of refinements in between the two in a coherent manner. Now, each new (lower) level in the hierarchy consists of subproblem abstractions that are less like global routing problems and more like detailed routing problems until, at the leaf nodes of the hierarchy, the routing problem becomes a detailed routing problem. Integrating both global routing and detail routing within the partitioning process also closely couples the method of routing the final subproblems to the decomposition. In this approach, the partitioning is continued until the subproblems may be trivially routed by case analysis or by a straight-forward maze routing step.

Also, a rip-up and re-route technique is investigated that may be applied to a hierarchy of routing subproblems, both at the detailed, leaf partitions as well as at higher, less-precise levels of the hierarchy. That is, the rip-up and re-route strategy uses the hierarchy in a bottom-up fashion.

Chapter 2

Hierarchical Decomposition

Hierarchical decomposition is a recursive divide-and-conquer method [AHU74]. Divide-and-conquer techniques split the original problem into smaller subproblems, solve the subproblems, and then recombine the partial solutions to form the solution to the original problem. In hierarchical decomposition, the problem is divided into subproblems that have the same form as the initial problem. Now, the subproblems are solved by recursively dividing each of the subproblems using the same process that was used for the original problem. The recursion is continued until some stopping criteria is met, and then the final subproblems are solved and recombined to form the solution to the desired problem. To apply hierarchical decomposition to general area routing it is necessary to decide how to partition the initial problem into subproblems, when to stop partitioning, and how to route the final subproblems, taking into account any interactions among them (i.e., boundary conditions.)

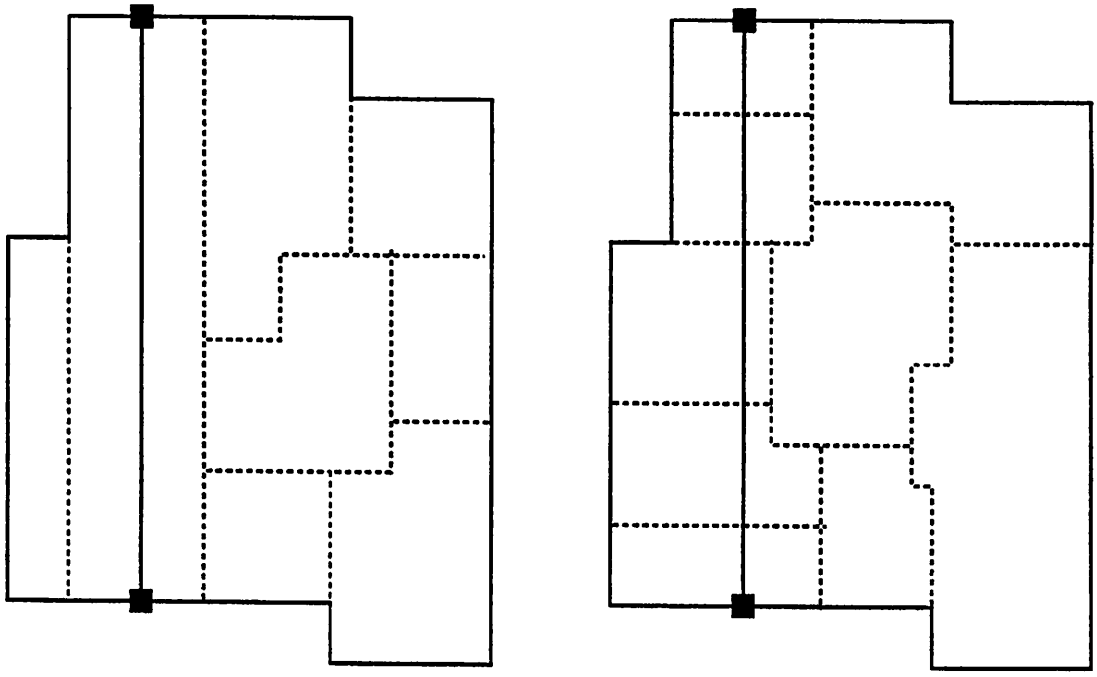
2.1 Dividing

In general, partitioning a routing problem into subproblems requires partitioning both the routing region and the nets of the problem. The routing region is partitioned by splitting the original region into subregions. Partitioning the nets involves splitting each net into sets of subnets such that each set is associated with one of the subregions. This requires determining if a particular net crosses a given subregion boundary and if it does, how many times it crosses and where it crosses. Thus, the formulation of a subproblem requires the specification of a corresponding region and set of nets just as in the specification of the

original problem. The concern here is to determine an optimal method for performing this decomposition in order to find a feasible solution to the original problem in a reasonable amount of time.

The goal of the partitioning process is to produce feasible subproblems that have complexity less than or equal to the original problem, since if the method is to be effective, the time required to partition a problem completely, to solve all its subproblems, and to recombine all the partial solutions must be less than the time required to solve the original problem and the collection of partial solutions must provide results that are equal in quality to a direct solution of the original problem. The definition of routing complexity in this situation involves both size and topology, and the method selected to perform the partitioning should account for both aspects.

The size complexity of a routing problem can be measured in many different ways. Size could refer to the area of the routing region, the number of edges in the geometry representing the routing region, the number of blockages, the number of nets, the number of pins, etc. Since part of the decomposition requires splitting up a routing region into subregions, it seems that reducing the size complexity of the problem as measured by its area is desirable. Thus, a partitioning method that divides the routing region into subregions of equal areas would be optimal in the sense that it decreases the area complexity of the original problem at a maximum rate and minimizes the height of the tree of hierarchical subproblems. Unfortunately, creating subproblems based on subregions of equal area does not necessarily lead to the simplest subproblems. In fact, problems that cover larger areas are not necessarily more complex than problems that cover smaller areas. Consider a portion of a routing problem that may be solved by connecting two pins with a single, straight metal line that spans the length of the chip. In this situation, the corresponding subproblem should be isolated as soon as possible and solved by specifying a single piece of metal instead of dicing it up and solving many smaller subproblems. An example of this is shown in Figure 2.1. Here, a particular net is routed by by a single straight path. In Figure 2.1(a), the region is decomposed so that the net may be routed in a single subproblem. In Figure 2.1(b), the region is decomposed so that the net is routed in several subproblems. The solutions to each of the subproblems are little, straight pieces of metal that must be recombined into one single path. When possible, the more efficient decomposition of Figure 2.1(a) should be used. Clearly, area complexity is only one factor that affects the complexity of the routing problem. Other kinds of size metrics such as number of nets and number of



(a) isolated net

(b) partitioned net

decomposition lines

net

Figure 2.1: Isolating a simple subproblem

obstacles also need to be considered.

The complexity of a routing problem also depends on its topology. Here, topology means the location (in all three dimensions) of the pins, the blockages, and the boundaries. Clearly, different topologies will admit a different number of feasible solutions. Assuming that having more possible solutions is less complex, different topologies have different complexities. Unfortunately, it is impossible to rank these different complexities exactly with out actually solving the problems and finding all the possible solutions. Thus, heuristics must be used to determine the relative complexity of different problems and these heuristics should consider both size and topological characteristics.

In order to determine how to simplify the routing complexity of the subproblems, the complexity characteristics of the original problem need to be considered. All non-trivial routing problems have at least one critical area where the number of permutations of routes that will result in a feasible solution for the whole problem is very small relative to other areas. If the net connections through these areas are not specified properly, then there is little hope of producing a complete routing of the entire problem.

Two views are proposed on how to handle these difficult regions in routing problems. One method is to choose a partition of the routing region that cuts through these critical areas on the basis that solving the hard sections first results in subproblems that are easier. The "solution" is provided by the pin assignment algorithm which determines the location of routes through these areas by assigning pseudo-pins to positions along the partition boundary[1]. Since the areas along this critical partition represent the most constrained portions of the routing problem, the resulting subproblems are less constrained and a pin assignment that satisfies the constraints on nets crossing this critical cut path is a necessary condition for the existence of feasible solutions to each of the subproblems. Moreover, since the hard parts of the original routing problem have been "solved", the subproblems should be easier to solve.

The opposing view is that a partition should be chosen that cuts through the non-critical areas of the routing region on the basis that such partitions minimally disturb the resulting subproblems. Each subproblem is no harder than the original problem with respect to critical areas. On the other hand, if a partition is made through a critical area

[1]This process is also hierarchical. The assignment at higher levels is to segments of the cut, rather than to specific locations. As the partitioning proceeds, the assignment becomes increasingly specific, until a fixed location is assigned.

and a poor solution is obtained from the pin assignment, then it will be hard to find a feasible solution for each of the subproblems. Since the critical areas have a small number of solutions that result in feasible solutions for the entire problem, it is very likely that a solution obtained for a critical cut path will be poor with respect to the original problem. This view assumes that critical areas are easier to solve in smaller problems. The argument for this assumption is that the smaller problems will have relatively smaller solution spaces in which to search for the solutions to these critical areas. Both approaches are investigated in this work.

In most of the previous work, straight lines were used to partition the routing region. These lines were either derived from some arbitrary routing grid [BP83b, MS84, HM85, Lau87] or based on the slicing structure of the floorplan [LTW86]. Often, the locations of the cut lines were chosen to bisect or quadrisect the region. Marek-Sadowska [MS86] uses tile planes to represent the routing region and allows cut lines that "zig-zag". Lauther [Lau87] reports that using a density- and capacity-based cut-line selection heuristic produces better results than choosing bisecting cut-lines. He defines the *criticality* of a cut line as $D - C$ where D is the density of the cut and is equal to the number of nets that must cross the cut and C is the total capacity of the cut. This definition of criticality corresponds to the philosophy of cutting through the most critical areas of the routing region. The trend towards more intelligent partitioning is continued in this work. Chapter 3 describes two methods of generating rectilinear cut paths. The first uses a graph theoretic algorithm and the second uses a scan-line-based technique. The chapter also contains a comparison of several different cut path selection heuristics.

To better understand the significance of rectilinear cut paths and "intelligent" selection heuristics, consider the situation of two offset blockages. This is shown in Figure 2.2. If cut paths are restricted to straight line segments and the canonical bisection heuristic is applied, then line segment $\overline{BE}$ will be chosen. If some capacity heuristics are also applied, then either segment $\overline{AD}$ or $\overline{CF}$ will be chosen. Unfortunately, these cut lines do not accurately capture the capacity of the partition that they represent. There exists a bottle-neck between the two obstacles that is not represented by either cut line. This area is the circled region in the figure. It will be very hard if not impossible to connect pins assigned to the segments $\overline{GD}$, $\overline{HE}$, $\overline{BI}$, or $\overline{CJ}$ because of the close proximity of one of the obstacles. Actually, a rectilinear cut path as shown in Figure 2.3 is desired. The rectilinear cut path is a much closer representation of the actual partition capacity and captures the topology

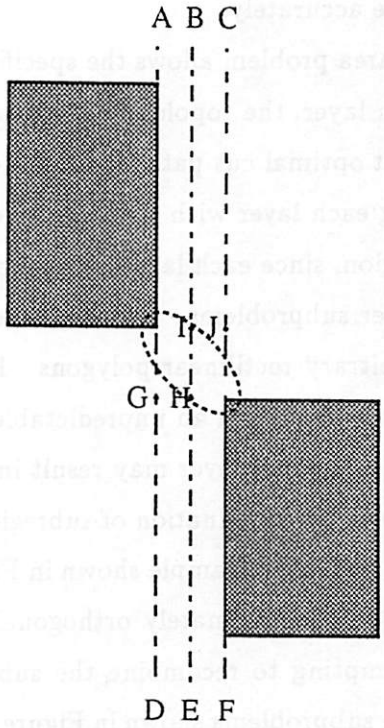


Figure 2.2: Possible straight cut paths near two blockages

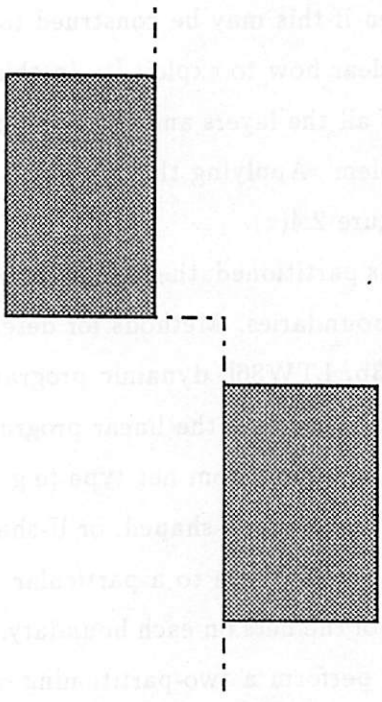
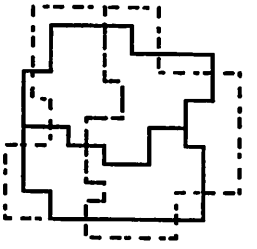


Figure 2.3: A possible rectilinear cut path

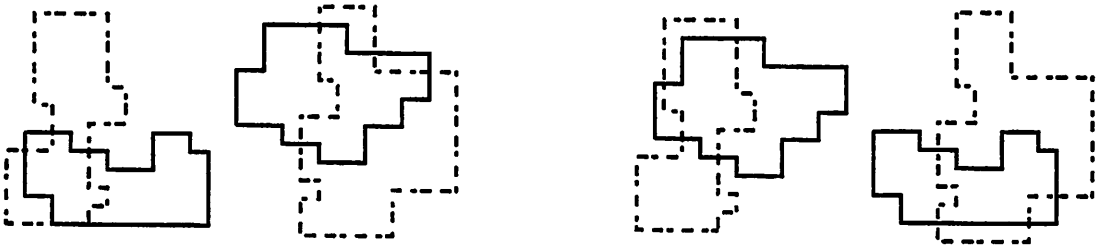
of the routing region more accurately.

Since a general area problem allows the specification of the routing boundary and obstacles to differ on each layer, the topology of each layer may be different. Accordingly, there may exist a different optimal cut path with respect to each layer. In some sense, one may consider partitioning each layer with its respective optimal cut path as the best way to partition the whole region, since each layer is optimally partitioned. Unfortunately, this does not guarantee simpler subproblems. On each layer, both the routing region and the routing blockages are arbitrary rectilinear polygons. Partitioning this geometry with an arbitrary rectilinear cut path results in an unpredictable number of subregions. Performing this partitioning on each individual layer may result in arbitrarily different subregions on each layer for which there is no combination of subregions that form a useful subproblem. Consider the pathological two-layer example shown in Figure 2.4(a). The cut paths selected for the individual layers are approximately orthogonal to each other. Partitioning each layer separately and attempting to recombine the subregions of the different layers into subproblems results in the subproblems shown in Figure 2.4(b). There are two possible ways of recombining the subregions into subproblems. It is ambiguous as to which combination to choose and in either case the subproblems are more irregular and more complex than the original problem. Even if this may be construed to be the optimal partitioning of the whole problem, it is not clear how to exploit it. In this work, cut paths are selected with respect to the topology of all the layers and the same cut path is used to partition all the layers of a particular problem. Applying this approach to the example might result in the subproblems shown in Figure 2.4(c).

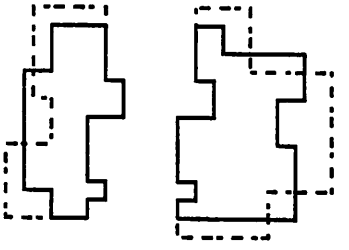
Once the region is partitioned, the nets must be partitioned and pseudo-pins must be assigned to subregion boundaries. Methods for determining these net crossings include linear programming [BP83b, LTW86], dynamic programming [BP83b, HM85], and linear assignment [MS86, Lau87]. Note that the linear programming and dynamic programming methods only provide the mapping from net type (e.g. 2-, 3-, or 4-terminal) to net route configuration (e.g. straight segment, L-shaped, or U-shaped). Heuristics are still needed to assign a particular net of a given type to a particular pattern and to specify the relative order and actual positions of the nets on each boundary. The linear assignment method has been used in routers that perform a two-partitioning of the routing region. Experiments based on a two-partition decomposition model and comparisons of linear assignment cost functions are described in Chapter 4.



(a) two layer problem and the cut paths found for each layer



(b) two possible ways of forming subproblems given the per layer cut paths in (a)



(c) subproblems formed using the same cut path for all layers

Figure 2.4: Partitioning per layer versus partitioning all layers

Another issue that arises in performing net partitioning is the number of times that a net is allowed or directed to cross a boundary. In [BP83b], a restriction is preferred that disallows net route configurations that cross a particular boundary multiple times because the higher levels of the hierarchy are routed under the assumption that this would not subsequently occur. In routers using two-partitioning, no such assumption is made and improvements have been reported when multiple crossings are selectively allowed. Lauther [Lau87] uses the crossing edges of the minimum spanning tree and Marek-Sadowska [MS86] uses a pin clustering method to determine the number of net crossings of a given net. Results for a minimum spanning tree method are given in Chapter 4.

2.2 Conquering

At some point the subdivision of the problem must stop and the solution of the subproblems must begin. The decision on when to stop partitioning is intimately related to the method that will be used to solve the subproblems.

In divide-and-conquer methods, it is often the case that the solutions to the subproblems are trivial and readily fall out of the partitioning process. This is the case for the hierarchical routers that use grid graphs. At the final level of the hierarchy, the grid graphs correspond to the actual routing grid and the solutions at this level are the solutions to the final subproblems. For detail routers, the routing grid represents the actual path positions. However, by definition and regardless of the use of grid graphs, hierarchical global routers stop at a higher level of abstraction than actual path positions. The particular level of abstraction may correspond to a channel assignment or a routing based on a gate-array cell grid. A detail router must then be used to complete the routing of the design.

Subproblem independence is another issue related to finding all the partial solutions. If the positions of the pseudo-pins on the partition boundaries are fixed and their layer assignments are determined then all the subproblems are independent. This is a nice property because it makes the solution of the whole problem independent of the order in which the final subproblems are solved. However, in this work, the pseudo-pins are allowed to float along their assigned partition boundaries and the assignment of specific layers to each of the pseudo-pins is delayed until the end. This is consistent with the concept of gradually refining the routing solution. Allowing these extra degrees of freedom makes all the subproblems dependent on their neighbors and an optimal ordering of the subproblems

must be determined. Experiments on finding a optimal routing region order are presented in Chapter 5.

In this work, hierarchical decomposition is used as a way combine both global and detail routing. For this method to be the most effective, the sum of the time required to completely partition a subproblem, solve the final subproblems, and recombine the partial solutions must be less than the time required to solve the original subproblem directly. In addition to comparing solution time, the issue of passing bad global decisions down the subproblem hierarchy must also be considered. If the final subproblems are too small, the detrimental influence of the previous bad decisions can not be avoided and a feasible solution will not be found. On the other hand, if the final subproblems are made larger, then bad decisions may only make them harder to solve instead of making them impossible to solve and a feasible solution will still exist. In other words, there exists some point of diminishing returns at which further partitioning is disadvantageous. Experiments using both a pattern router and a maze router to route the final subproblems are described in Chapter 5. The pattern router corresponds to complete partitioning as in [BP83b] and the maze router corresponds to partial partitioning.

2.3 Summary

Hierarchical decomposition can be viewed as a means of gradually refining the routing solution from a rough approximation to an exact solution. In this sense, each level of the hierarchy can be seen as a point on a continuum between global and detail routing. This is seen more clearly if the assignment of pseudo-pins to partitions is interpreted as effecting a global route. Assigning a pseudo-pin to either a specific position or a range of positions on a partition provides a means of transmitting global information and constraints to the subproblems. The assignment represents a narrowing of the search space of feasible routing solutions and thus represents a global routing of the nets.

In the following chapters, experiments on the different aspects of hierarchical decomposition are discussed in more detail.

Chapter 3

Region Decomposition

The first step in dividing a routing problem into subproblems is to divide the routing region into subregions. This requires choosing a line or path along which to partition the original routing region. Historically, cut lines have been chosen as straight lines that approximately bisect the region. These experiments continue the trend towards more complex cut path selection heuristics. Two major methods of selecting cut paths have been tried. The first is based on capturing the topology of the routing region in an acyclic directed graph and the second is a scan-line-based technique. Within each method, experiments have been performed with a variety of cost functions. For simplicity, a single cut path is selected for each subproblem and a two-partitioning of the region is performed. However, these selection methods can be easily extended to perform a more general four-partition. In fact, the two-partition can be thought of as a step in producing a four-partition and thus the results on cut path selection heuristics should also hold for a four-partition paradigm.

3.1 Graphs

The goal of choosing "intelligent" cut paths is to choose those paths that will partition a routing problem into subproblems that are less complex than the original. Thus, the selection method must be capable of representing and analyzing the topology of the routing region. Furthermore, the chosen partition must accurately reflect the routing capacity of the common subregion boundary that it represents so that it can supply the subsequent net decomposition with accurate information. The first method investigated captures the topology of the routing region in a constraint graph. That is, an acyclic directed graph that

represents the routing region is generated such that the edges of the graph correspond to edges of the routing region, edges of the routing obstacles, and potential bottlenecks in the region. The constraints of the routing problem are transformed into edge weights and then the problem of finding a cut path is formulated as a shortest path problem. The relationship between the graph and the routing region is described in more detail below.

The real work in generating the graph representation of a routing region is determining the graph edges that will represent the routing bottlenecks. Generating the other edges is relatively easy since the edges of the graph, corresponding to edges of the routing region and edges of the routing obstacles, map one-to-one with segments or subsegments of the corresponding input geometry. To get the bottleneck edges, rays are extended orthogonally from vertices of the routing region and the routing blockages into the routing region until they hit other blockages or boundaries. The resulting line segments become bottleneck edges. Certain diagonal edges are also introduced, corresponding to bottlenecks not represented by the other edges. The nodes of the graph correspond to intersections of certain orthogonal segments. In particular, the intersections of routing region edges and obstacle edges with other routing region edges, other obstacle edges, and bottleneck edges transform to nodes in the graph, but intersections between bottleneck edges do not create nodes. This representation is defined more formally in the following.

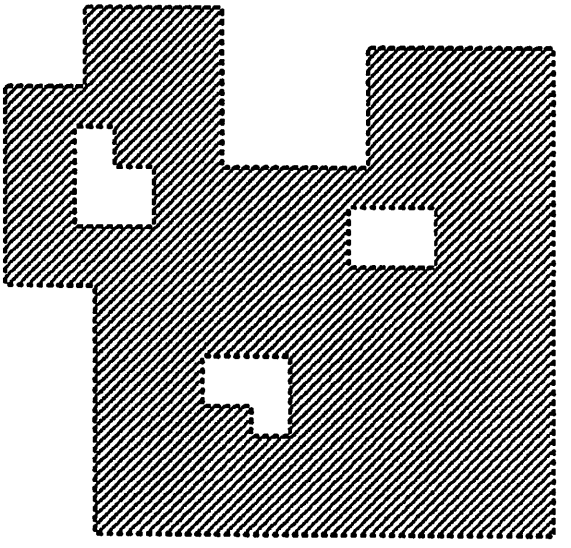
Consider the geometry representing the routing region. This geometry consists of a minimal set of nonoverlapping Manhattan polygons on each layer. The routing area corresponds to the presence of geometry and routing obstacles, along with the routing boundaries, correspond to the absence of geometry or to holes in the geometry. Figure 3.1(a) shows an example one-layer routing region and Figure 3.1(b) shows an example two-layer routing region. Let the coordinates of a point, P, be x_P, y_P.

$$P = (x_P, y_P)$$

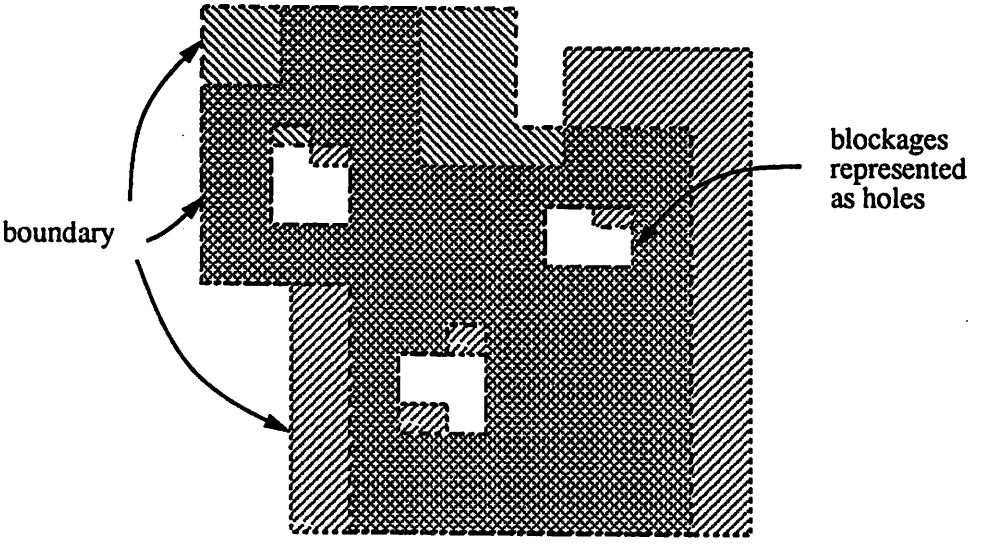
Let P_i^G and L_i^G be respectively the set of vertices and the set of edges of the geometry on routing layer i. The superscript G denotes that the points and edges come from the original geometry.

$$\begin{aligned} P_i^G &= \{(x,y) \mid (x,y) \text{ a vertex of the geometry on layer } i\} \\ L_i^G &= \{\overline{AB} \mid \overline{AB} \text{ a line segment of the geometry on layer } i\} \end{aligned}$$

To represent the bottlenecks between different blockages and between blockages and the



(a) single layer routing region



(b) two-layer routing region

Figure 3.1: Example routing region geometry

routing region boundary, create line segments that span these areas and let the line segments correspond to edges. The first set of such segments consists of horizontal and vertical segments that span a section of the geometry between a vertex and another vertex or between a vertex and an edge. The path of each segment can be thought of as starting at either a boundary corner or obstacle corner and extending into the routing region until hitting another blockage of the same layer or the border. Here the border is defined as the bounding Manhattan polygon of the whole routing region. Let R^G be the edges of the border of the routing region and let L_i^C be the set of line segments constructed by extending rays from the vertices on layer i until they intersect an edge or vertex of the same layer or of the routing region border. First define

$$L_i^H = \{\overline{AB} \mid A \in P_i^G, \overline{AB} \notin L_i^G, y_A = y_B, \overline{MBN} \in L_i^G, R^G, |\overline{AB}| = \overset{min}{B} |x_A - x_B|\}$$

and

$$L_i^V = \{\overline{AB} \mid A \in P_i^G, \overline{AB} \notin L_i^G, x_A = x_B, \overline{MBN} \in L_i^G, R^G, |\overline{AB}| = \overset{min}{B} |y_A - y_B|\}$$

where L_i^H and L_i^V are the segments constructed by extending a ray horizontally or vertically, respectively. Then,

$$L_i^C = L_i^H \cup L_i^V$$

where L_i^C is the set of *constructed*, orthogonal line segments. Figures 3.2 and 3.3 show the sets L_i^H, L_i^V, and L_i^C for $i = 1, 2$ corresponding to the example of Figure 3.1(b). For reference, Figures 3.2(a) and 3.3(a) show layers 1 and 2, respectively. Figures 3.2(b) and 3.3(b) show the constructed horizontal segments on the left and the constructed vertical segments on the right. The constructed segments are shown as solid black lines. Finally, Figures 3.2c and 3.3c show the union of the constructed horizontal and vertical segment sets. In parts (b) and (c) of Figures 3.2 and 3.3, outlines of both layers are shown to indicate the full extent of the routing region.

The nodes of the graph correspond to the vertices of the original geometry and to the intersections of constructed segments with original geometry segments independent of layer. First define the layer independent sets,

$$L^C = \bigcup_i L_i^C,$$
$$L^G = \bigcup_i L_i^G$$

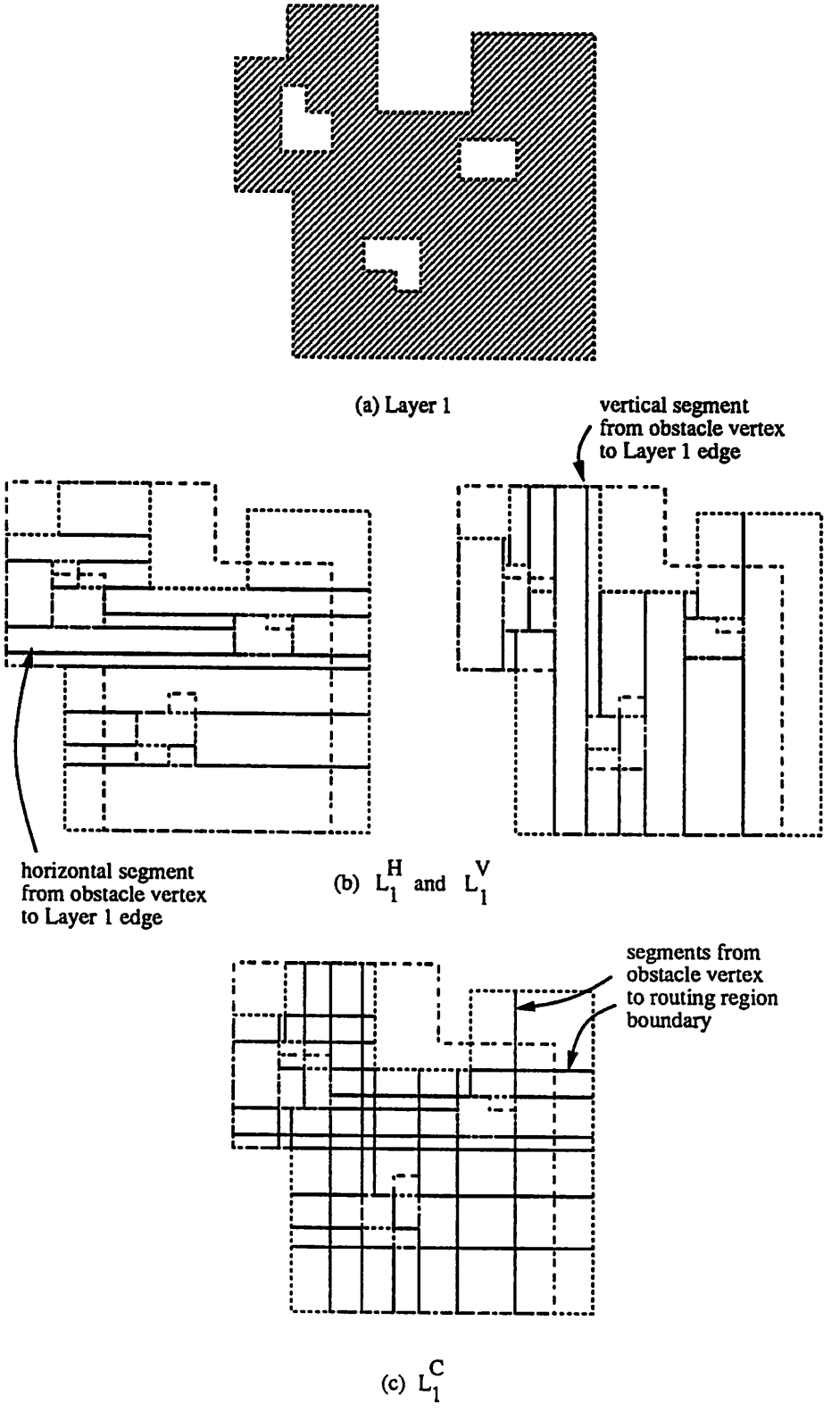


Figure 3.2: Line segment construction for layer 1

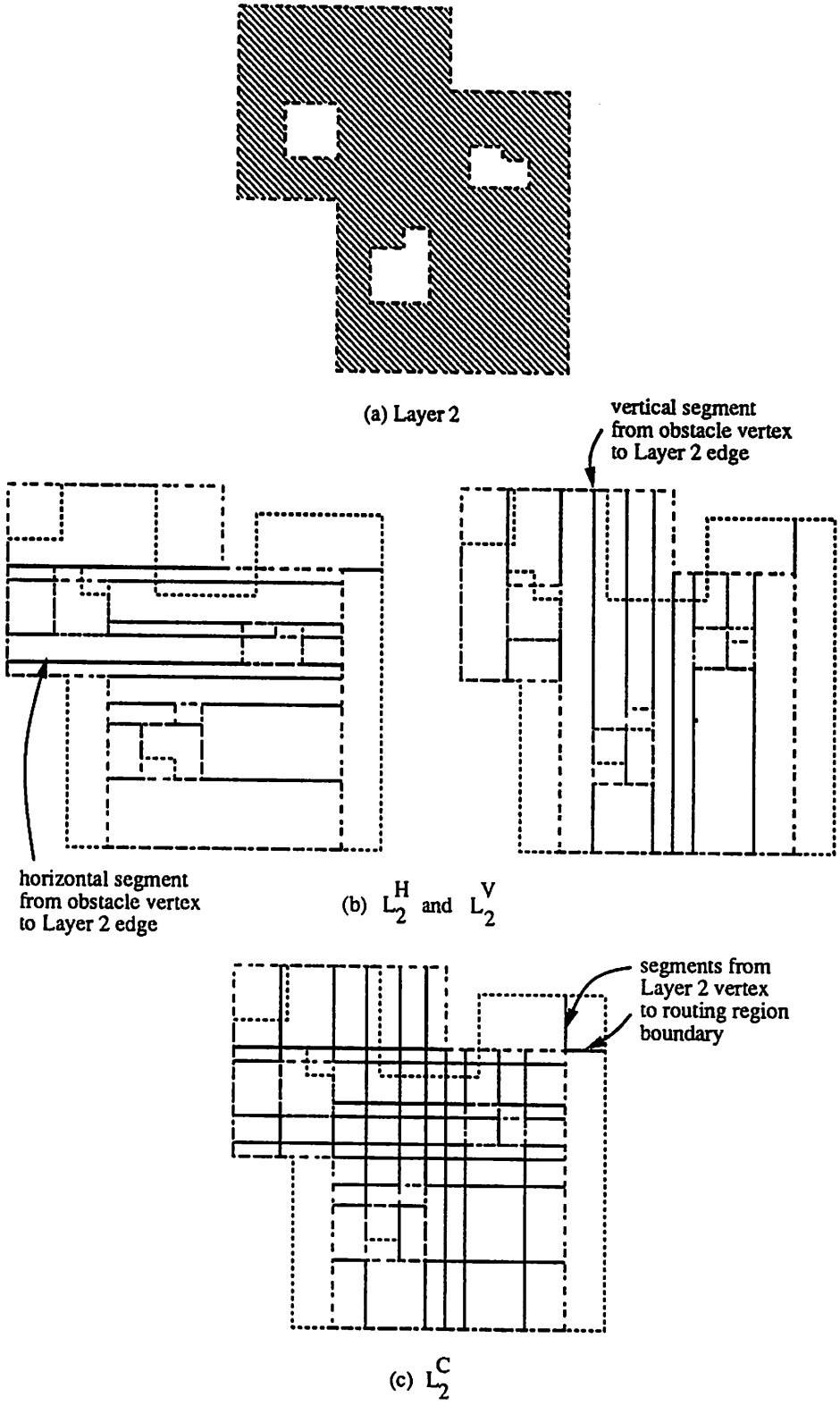


Figure 3.3: Line segment construction for layer 2

and

$$P^G = \bigcup_i P_i^G.$$

These sets are the set of all constructed segments, the set of all original geometry segments, and the set of all original geometry vertices, respectively. The set of intersections between constructed segments and original geometry segments is the set of points

$$P^C = \{X \mid \exists\, \overline{AXB} \in L^G, \exists\, \overline{CXD} \in L^C\}.$$

Thus, the complete set of points, V, that corresponds to nodes in the graph is

$$V = P^G \cup P^C.$$

The set V for the example of Figure 3.1 is shown as solid black dots in Figure 3.4.

The edges of the graph correspond to subsegments of both the constructed and original geometry segments whose endpoints correspond to nodes in the graph. If the universe of segments is denoted by

$$L = L^G \cup L^C$$

then the set of segments corresponding to edges in the graph is

$$L^E = \{\overline{AB} \mid A, B \in V, \overline{MABN} \in L\}.$$

Some of the bottlenecks in the routing region can not be represented by a Manhattan line segment. To capture these features in the graph, certain edges are created that correspond to line segments that are diagonal with respect to the original geometry. The segments considered are the set of diagonal segments that span the geometry between obstacles and between obstacles and the routing region boundary. These segments have endpoints which are vertices of the original geometry and do not intersect any segments of the original geometry except at their endpoints. In order to make the resulting graph acyclic, only the subset of these segments that have positive slope are considered. This set of candidate segments is given as

$$D = \{\overline{AB} \mid x_A > x_B, y_A > y_B, \overline{AB} \cap L^G = \{A, B\}, \overline{AB} \cap \text{ geometry } = \overline{AB}\}.$$

These candidate segments can be found using a scan-line procedure that maintains a history list of possible diagonal segment endpoints. The subset of D that will have corresponding

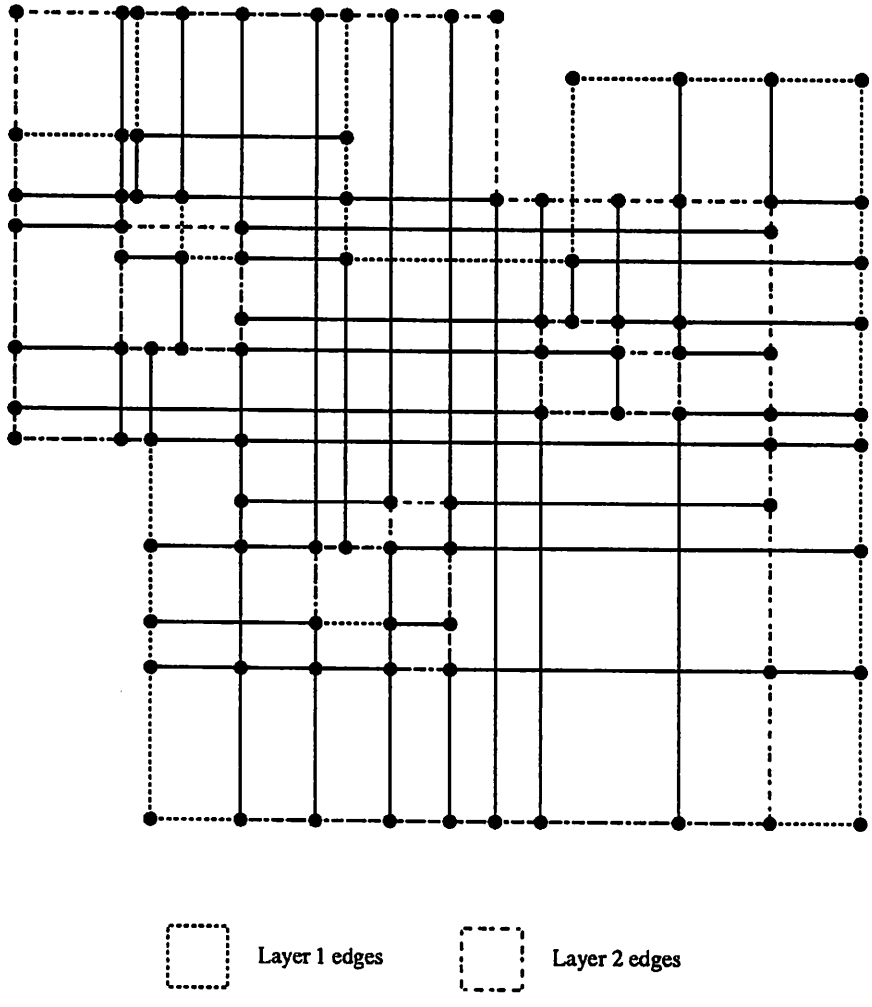


Figure 3.4: Nodes of the two-layer example

edges in the final graph are those segments that would create an edge for which a path corresponding to Manhattan segments does not already exist. If a path corresponding to Manhattan segments already exists, then the corresponding diagonal edge is redundant and unnecessary. Though these redundant edges do not make any difference to the shortest path algorithm, diagonal edges must be treated specially by the graph building and maintenance routines, and since the ability to ignore these redundant diagonal edges comes for free in the graph building algorithm, the redundant edges are not included in the graph representation. Here, D has been defined as a superset for clarity and ease of specification. The chosen subset of D can be described more accurately in terms of an intermediate graph construction explained later. Figure 3.5 shows the subset of D that would be chosen for the two-layer example. In this particular case, it is a single line segment near the middle of the routing region.

Now, the graph representation of the routing region may be formulated by defining the mapping from points and line segments to nodes and directed edges. For the sets of points and segments defined above, this mapping is one-to-one. The direction of edges is chosen so that if the corresponding line segments are considered, vertical line segments are directed down, horizontal line segments are directed left, and diagonal line segments are directed down and left. This mapping may be defined as

$$
\begin{aligned}
P &\mapsto \text{node } p \\
\overline{AB} &\mapsto \begin{cases}
 & \text{if} \quad x_A = x_B \text{ and } y_B < y_A \\
(a,b) & \text{or} \quad y_A = y_B \text{ and } x_B < x_A \\
 & \text{or} \quad x_B < x_A \text{ and } y_B < y_A \\
(b,a) & \text{otherwise}
\end{cases}
\end{aligned}
$$

An initial graph of edges corresponding to Manhattan edges only is $G' = (N, A')$ where

$$
\begin{aligned}
N &= \{p \mid \exists P \in V s.t.\ P \mapsto p\} \\
A' &= \{(i,j) \mid \exists \overline{AB} \in L^E s.t.\ \overline{AB} \mapsto (i,j)\}
\end{aligned}
$$

To this initial graph, edges, corresponding to diagonal segments, are added only if the addition of the edges create new paths in the graph. This set of edges is

$$D' = \{(i,j) \mid \exists \overline{AB} \in D\, s.t.\ \overline{AB} \mapsto (i,j), \nexists \text{ a directed path in } G' \text{ from } i \text{ to } j\}.$$

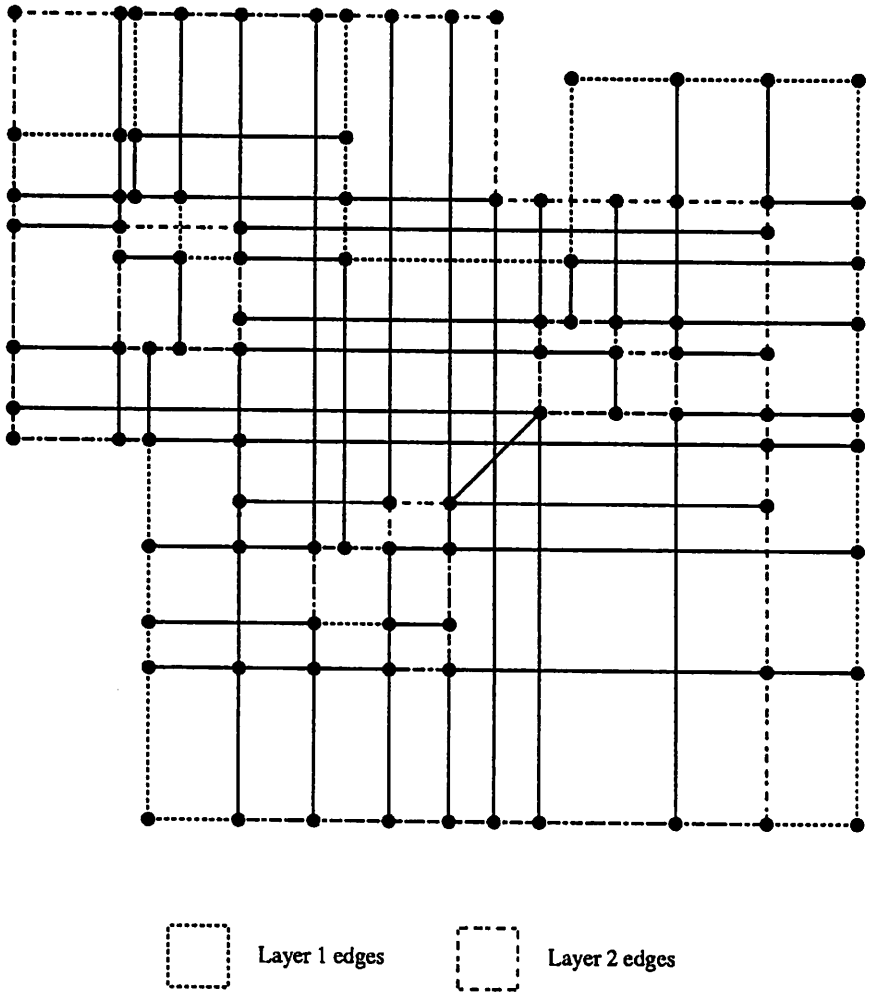


Figure 3.5: Two-layer example with diagonal line segment added

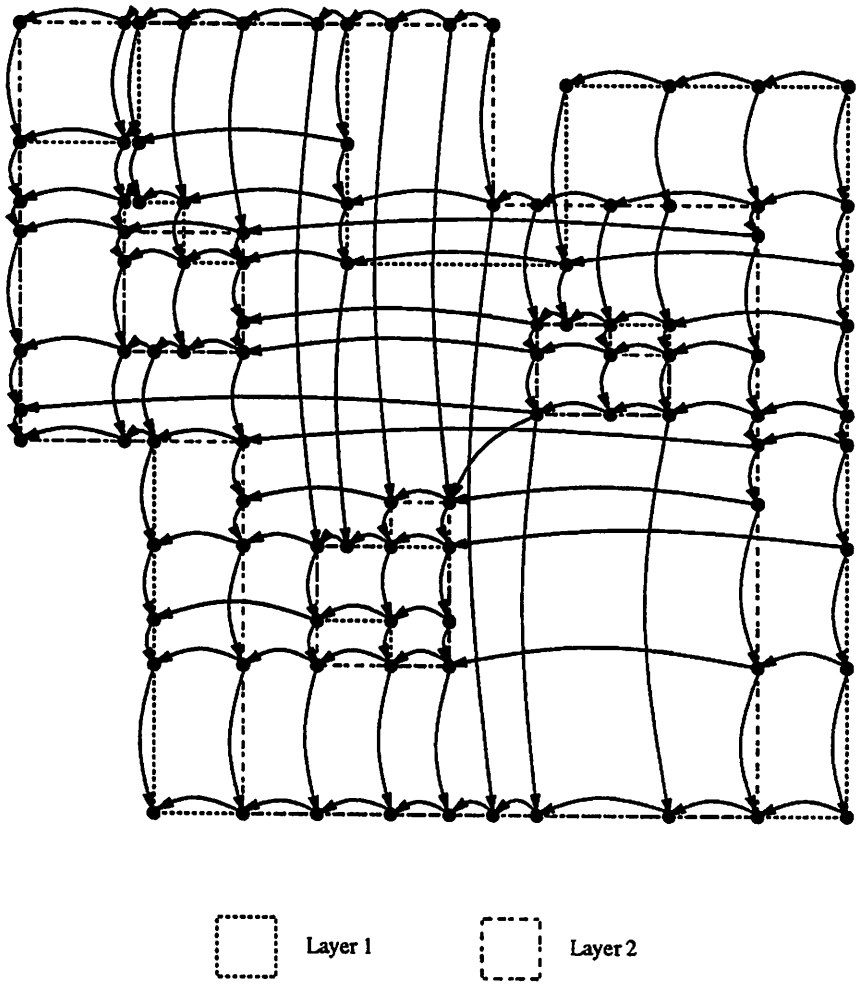


Figure 3.6: Graph corresponding to two-layer example

Now let

$$A = A' \cup D',$$

then the basic graph used to represent the topology of the routing region is

$$G = (N, A).$$

Figure 3.6 gives the graph corresponding to the example two-layer routing region.

The problem of finding a cut path for a routing region can be formulated as a shortest path problem on the graph representing the routing region. Weights are assigned to the edges of the graph depending on the length and capacity of their corresponding line segments. Note that the capacity along the length of each segment is homogeneous. The

shortest path with respect to these weights is found and transformed back to its corresponding line segments. These segments form the rectilinear path that is used to partition the routing region. Since the path is required to be Manhattan, diagonal line segments are arbitrarily converted into two Manhattan line segments such that the resulting corner is concave up.

To solve the shortest path problem on this graph, the graph first must be massaged into a canonical form. The desired form is that of a polar acyclic digraph. Let a *border edge* be defined as an edge of G that corresponds to a line segment that is coincident with the routing region boundary and similarly, let a *border node* be defined as a node of G that corresponds to a point that is coincident with the routing region boundary. Here, the routing region boundary is defined by the bounding Manhattan polygon of all the input geometry. Note that with respect to partitioning the routing region, a path, A, that contains border edges is equivalent to the path, B, that results from removing all border edges from path A. Thus, the border edges need not be considered when looking for the shortest path. In the following, it is assumed that all border edges have been deleted from G and that any nodes that have zero degree as a result of this operation are also deleted. To get the graph into the desired form, dummy source and sink nodes are added and then connected to selected border nodes. Border nodes with only fan-out edges are connected to the source node by edges directed from the source. Border nodes with only fan-in edges are connected to the sink node by edges directed to the sink. Border nodes with both a fan-in and fan-out edge are connected to either the source or the sink depending on the desired cut path orientation. This orientation may be either horizontal or vertical and alternates at each level of partitioning. Of the border nodes with both a fan-in edge and a fan-out edge, the nodes with fan-in edges corresponding to segments with the same orientation as the desired cut path orientation are attached to the sink by edges directed to the sink. The remaining border nodes with both a fan-in edge and a fan-out edge are attached to the source by edges directed from the source. The canonical graph for the two-layer example is shown in Figure 3.7. The border nodes are connected to the source and sink so as to favor horizontal cut paths. Nodes enclosed in a solid box are connected to the source by a directed edge from the source and nodes enclosed by a dot-dash box are connected to the sink by a directed edge to the sink.

After the above modifications, the shortest path in the graph can be found using standard algorithms. If the nodes of the graph are sorted topologically, the shortest path

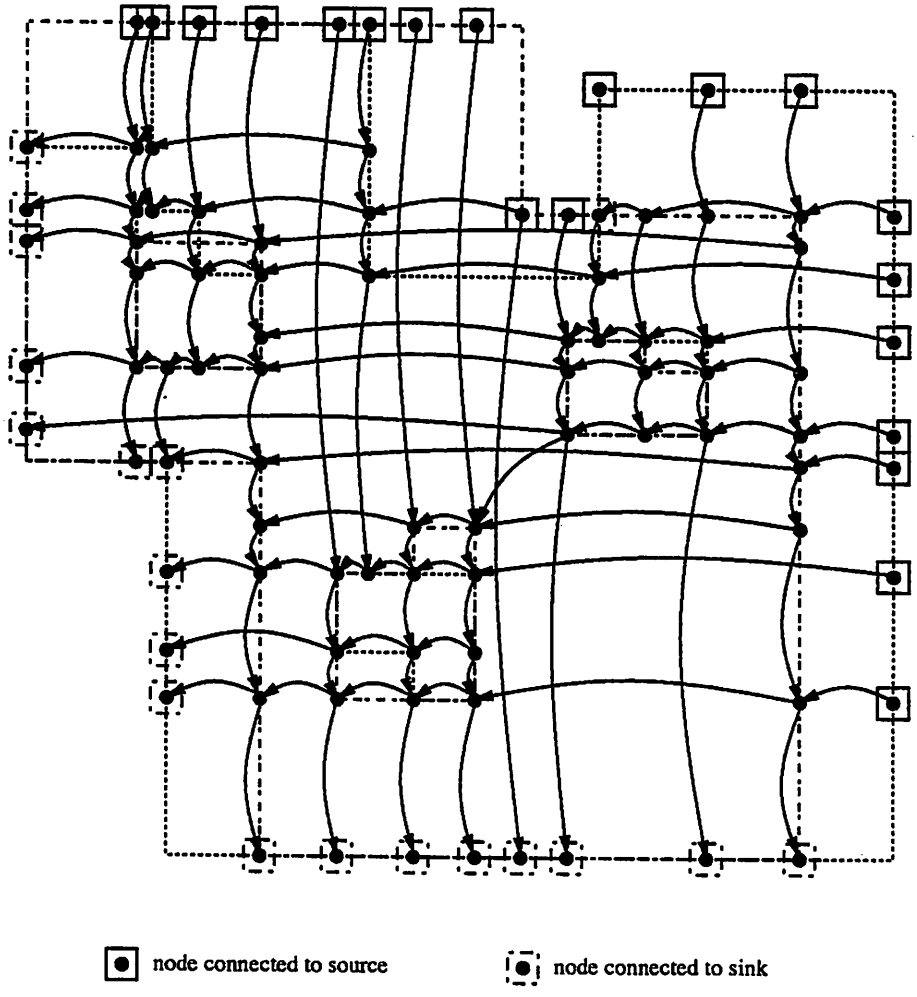


Figure 3.7: Graph for solving the shortest path problem

may be found in $O(m)$ time where m equals the number of edges. The topological sort of the graph also takes $O(m)$ time. Therefore, the shortest path and equivalently a cut path can be found in $O(m)$ time. For a complete graph, $m = n^2$ where n equals the number of the nodes. Fortunately, the graph representation defined above is very sparse. Excluding diagonal edges and edges connecting to the source and sink, the maximum number of fan-in edges (or equivalently, fan-out edges) per node is two. So, in practice, m is very close to $2n$ and the time complexity of this shortest path algorithm is close to $O(n)$.

Different variations of this graph were tried in attempts to reduce graph building overhead at each level of the hierarchy and to improve the correspondence between the shortest path and the desired cut path. Changes to the graphs include making the graph planar so that the graph of a subproblem corresponds to a subgraph of the graph of the original problem and removing nodes and/or edges to reduce both the size of the problem and the number of possible "bad" paths that can be found.

As described above, the basic graph is not planar. This means that after partitioning, the graph of a subregion may not correspond to any subgraph of the original graph and a new graph must be constructed for each subproblem. Unfortunately, this duplicates much of the previous computation, since a large percentage of the new graph will be the same as a portion of the original graph. Differences will occur where an edge in the original graph crossed the cut path. Having a planar graph representation would eliminate this extra work since the original graph could be partitioned along with the routing region to produce the appropriate subgraphs for the subregions. This would also allow the use of data on the graph to transmit global information through out the hierarchical decomposition. The simplest way to transform the basic graph into a planar graph is to flatten it into the plane. That is, add to the graph the nodes corresponding to the intersections of the line segments in L^E and divide the old edges into new edges corresponding to the set of non-overlapping subsegments induced by the new node set. While this produces the desired planar graph, it has the problem of greatly increasing the size of the graph and admitting more paths in the graph that do not correspond well with potential desired cut paths. Table 3.1 shows some data comparing the size of the basic graph with the size of a "flattened", planar basic graph. To see how flattening the basic graph admits more "bad" paths, consider the portion of a routing region and its corresponding planar graph as shown in Figure 3.8. The nodes marked by X's are induced by the surrounding blockages, but do not represent directly any useful topological information. Most of the extra paths admitted by these nodes are

Examples	Nonplanar Graph		Planar Graph	
	Nodes	Edges	Nodes	Edges
adder2	859	1780	1335	2732
adder4	1762	3636	3620	7349
adder32	13047	26463	35146	70655

Table 3.1: Number of nodes and edges in planar versus nonplanar graphs

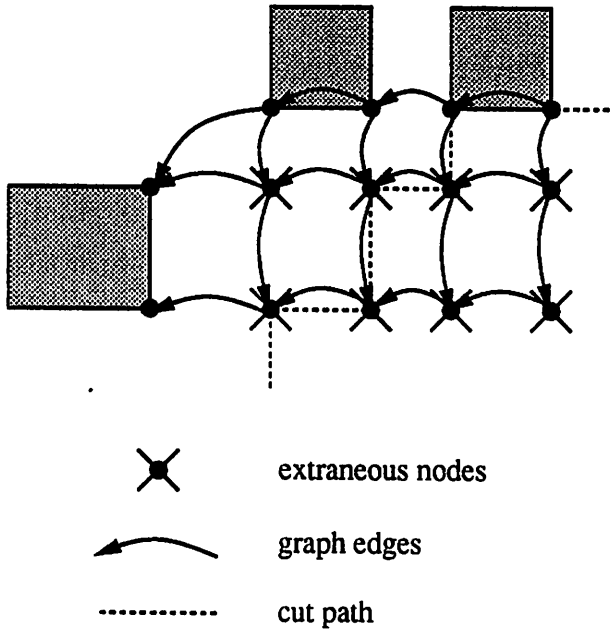
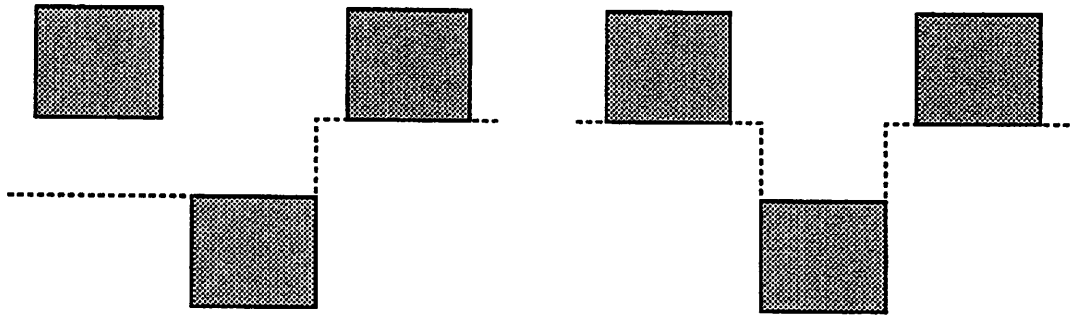


Figure 3.8: A bad path in a planar graph



(a) monotonic non-decreasing path (b) non-monotonic path

Figure 3.9: The need for non-monotonic cut paths

undesirable, and pathological cases can arise easily that will allow these "bad" paths to be selected. An example of one such path is shown in the figure.

Another way to produce a planar graph representation of the routing region is to start with the basic graph and selectively remove edges to make the graph planar. Edges can be ranked to favor the removal of long edges that intersect many shorter edges. This heuristic is intended to remove any potentially undesirable paths from the graph first. While this alleviates the size problem of planar graphs, it now allows the possibility of removing the optimal cut path from the graph in the process of making the graph planar. The resultant graph also suffers from some generic problems that are described next.

Given the basic graph representation, all paths are monotonic non-decreasing from left to right. This means that given a situation like the one in Figure 3.9, the best cut path that is represented in the graph is shown in Figure 3.9(a). Unfortunately, the path shown in Figure 3.9(b) is preferable. On the examples, the graph algorithm generated cut paths that looked like stair cases and tended to slice slivers off the region. These kind of paths do little to simplify the resulting subproblems from the perspective of size and geometric complexity. In fact, these paths often result in subregions that are more complex than the original problem.

The fundamental problem with this approach is its formulation as a shortest path problem. Consider the heuristic to find cut paths that "hug" blockages and cross between blockages at the narrowest sections. These paths should represent potential critical bot-

tlenecks in the problem. The corresponding objective function is to find the path with minimum capacity relative to total length or more formally

$$\text{minimize } \frac{\sum_{ij} c_{ij}}{\sum_{ij} a_{ij}}$$

where c_{ij} is the capacity of edge (i,j) and a_{ij} is the length of edge (i,j). This is clearly a rational objective function. Other possible heuristics also correspond to rational objective functions. These heuristics include maximizing net density relative to total length and maximizing net density relative to total capacity. Also, heuristics to force the graph algorithm to find a cut path near the middle of the routing region require global rather than local information. That is, determining whether a path approximately bisects a region requires examining the entire path, and a bisecting path can not be constructed incrementally from local information. Thus, the problem of finding a cut path on the graphs described above has become a minimum ratio cycle problem and no longer a shortest path problem. Unfortunately, solving this problem is impractical because its time complexity is too high for the size of problems that will be encountered. The naive implementation is $O(m^2n^2)$ and the best known result is $O(mn^2log(n))$ using Megiddo's method [Meg79].

3.2 Scan-Lines

In general, partitioning a general area routing problem results in subproblems that are also general area problems. However, the experiments with constraint-graph cut-path selection methods showed that the general area subproblems could easily become more complex than the original problem. Thus, while arbitrary rectilinear cut paths are allowed, "straighter" cut paths will make life easier. That is, the following approach should be considered. Choose a partition from the perspective of starting with a straight cut line and then make a minimal number of perturbations to create a cut path with the desired topological characteristics. In the following, a scan-line method with *sticky* segment heuristics is introduced that implements this perspective.

The standard scan-line method begins by sorting either the vertices or the edges of the geometry in question. If the case of sweeping a vertical line horizontally across the region is considered, then the vertices will be sorted first by x-coordinate and then by y-coordinate. Each group of vertices with the same x-coordinate represents a different scan line. As the list of vertices is processed, a new scan line is formed as each new group of

vertices with common x-coordinate is encountered. In the proposed variation, portions of a scan line are allowed to stick to blockage edges as the scan line passes through the routing region. When a grouping of vertices triggers a new scan line, instead of automatically updating all sections of a scan line to reflect the new common coordinate, heuristics are applied to determine which scan line segments will move to the new coordinate position and which segments will *stick* to previous positions. At each coordinate level, the cost of the cut path corresponding to the current scan line is calculated and a record of the minimum cost path is maintained. To partition the original problem, scan line passes are made both vertically and horizontally and the best path over both orientations is selected. After that, subsequent levels in the hierarchy alternate orientations. Exceptions to this pattern occur if a path with the specified orientation does not exist. In these cases, a bisection of the region is attempted and if this is not possible, a path of the other orientation is used.

Figures 3.10(a) to 3.10(d) show the desired sequence of horizontal scan lines that will capture the potential critical areas of the three-obstacle case discussed earlier. After skipping over the routing region border, the first scan line is triggered by the beginning of the lower obstacle. The scan line is a straight line consisting of three segments. The middle segment has lower capacity than the other two segments because it corresponds to an edge of an obstacle. The next scan line is emitted due to the top of the lower obstacle. All segments of the scan line move to the new position since the corresponding capacity of the new segments are the same and a straight cut line is preferred to a cut path if it is equivalent. The next scan line is emitted when the upper two obstacles are encountered. In this case, the middle segment of the previous scan line sticks to the lower blockage because its corresponding new segment has a higher capacity. The final scan line is created in response to the top of the upper blockages. The jog in the previous scan line has snapped back to align with its neighboring segments because the capacity of the jog segments would exceed the capacity of the corresponding new straight segment.

The heuristics that are used to generate an appropriate sequence of scan lines are discussed in the following. For each set of vertex events that represent a new possible scan line position, it must be decided for each segment of the current cut path whether or not to move it to the new position. Here and in the following discussion, segment refers only to those segments of the cut path that are parallel to the scan line. With respect to the capacity of the corresponding new segment, there are three possible relationships. The new segment has capacity less than, equal to, or greater than the old segment. If the new

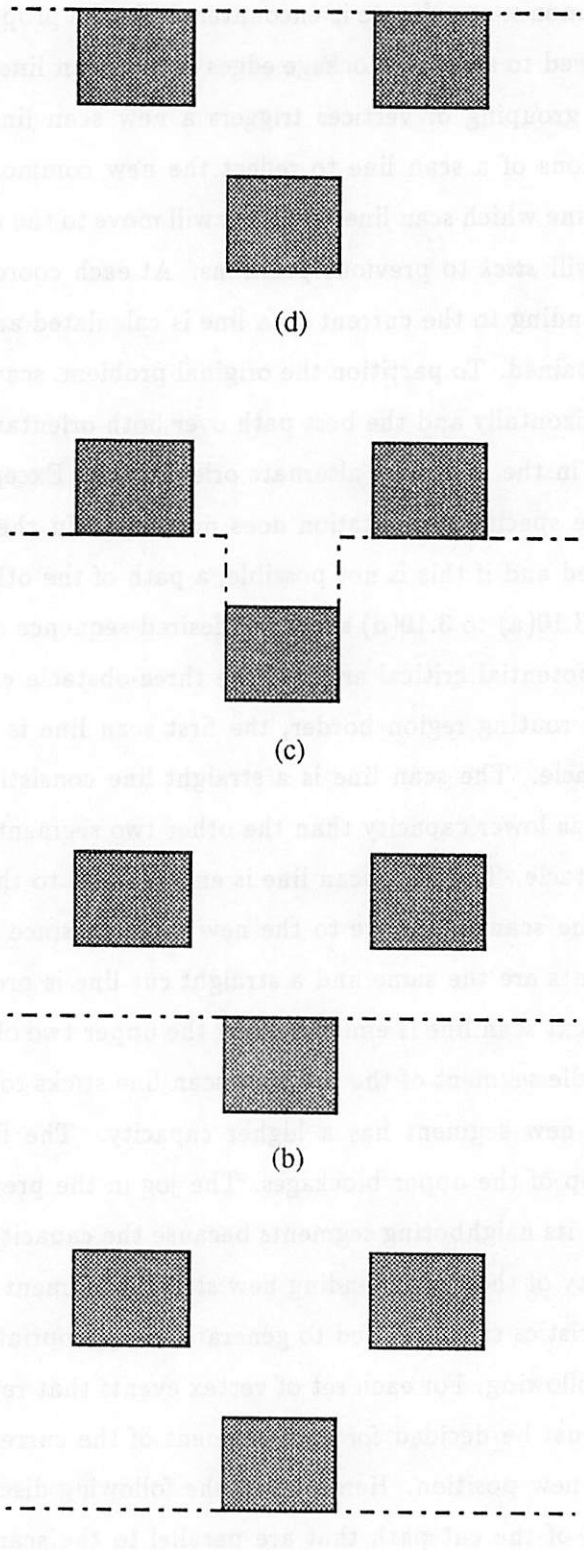


Figure 3.10: Scan-line sequence

segment will have capacity less than the capacity of the current segment, then the current segment is always moved. Changing to segments of lower capacity represents trying to hug the blockages. It also helps break the scan line away from cut paths of minimum local cost by forcing the cut path to advance across the routing region. For all other segments, the cost of the cut path resulting from changing to the new segment is compared to the cost of the unchanged cut path. If the cost does not increase, then the current segment is moved. This is consistent with the goal of minimizing the necessary deviations from a straight cut path. For any given state of the scan line, it may be possible that the optimal cut path can only be formed by first moving segments that increase the cost of the cut path. Thus, the generated cut paths are dependent on the order of segment comparisons. The following order heuristics are applied. Let Set 1 be the set of all segments that have corresponding new segments of lower capacity, Set 2 be the set of segments that have corresponding new segments of equal capacity, and Set 3 be the set of segments that have corresponding new segments of higher capacity. First, all segments Set 1 are moved. Next, the effect of changing each segment in Set 2 is analyzed. Any segment that will not increase the cost of the cut path is moved. Then, each segment in Set 3 is tested. Again, any segment that will not increase the cost of the cut path is moved. Finally, among the segments that have not been moved, the segments that are sets of contiguous, colinear segments from both Sets 2 and 3 are checked. These "mixed" segments are also moved if they will not increase the cost of the cut path. After all these cost comparisons, one more heuristic is applied to help prevent wildly irregular cut paths. All segments in the scan line that are not at the level of the new segments are checked. If such a segment forms a U-shaped bend with its neighboring segments, then the aspect ratio of the bend is checked. If the distance to its closest neighbor (in the direction orthogonal to the scan line) is greater than its length, then the segment is advanced to the level of the closest neighbor. That is, whenever the aspect ratio of a peak of a bend becomes greater than unity, the peak is flattened. This heuristic is applied repeatedly to the scan line until no segment meets the specified conditions.

This method improves on many of the problems of the constraint graph method. The cut paths are not constrained to have monotonic changes in direction, and the worst stair-case topologies are prevented because the method inherently connects opposite sides of the routing region. The cost function evaluates an entire path at a time and thus allows comparisons of rational objective functions. Also, the cost function can be easily modified to allow a cost component or factor that favors paths that approximately bisects the routing

region. This prevents paths that slice off slivers of the region and results in better balanced subproblem trees.

3.3 Cost Functions

The intelligence in either of the cut path selection methods discussed above lies in the cost functions. These heuristics provide a way of ranking cut paths as to their "criticalness". In these experiments, different definitions of "critical" have been compared and the difference between choosing the "most" critical paths versus choosing the "least" critical paths has been investigated.

Let P be a cut path consisting of edges or segments and let each edge, (i, j), have associated values a_{ij} and c_{ij} where

$$\begin{aligned} a_{ij} &= \text{the length of (i,j)} \\ c_{ij} &= \text{the capacity of (i,j)} \end{aligned}$$

and both are measured in number of routing tracks with respect to the standard two-layer wiring model.

Let the total length, L, of a cut path, P, equal the sum of the lengths of all of its segments,

$$L = \sum_{i,j \in P} a_{ij}$$

and let the total capacity, C, of a cut path, P, equal the sum of the the capacities of all its segments,

$$C = \sum_{i,j \in P} c_{ij}.$$

Let

$$D = \text{the net density of the cut path,} P.$$

Here the net density of a cut path is defined as the number of nets that cross the path. A net is assumed to cross a path if the bounding box of its pins intersects the path.

The following cost functions have been tried. In each case the cut path of minimum cost is chosen. Cost functions (2) through (6) correspond to choosing "critical" cut paths and cost functions (7) through (11) correspond to choosing "non-critical" cut paths.

(1) **Bisection**

This is a control case and not actually a cost function. A straight cut path that most nearly bisects the routing region is chosen.

(2) **Minimum Capacity**

$$cost = C$$

The minimum cost cut path corresponds to the minimum capacity cut path.

(3) **Maximum Density**

$$cost = -D$$

The minimum cost cut path corresponds to the cut path of maximum density.

(4) **Critical**

$$cost = C - D$$

The minimum cost cut path corresponds to the most "critical" cut path as defined by [Lau87].

(5) **Maximum Bottleneck**

$$cost = \frac{C}{L}$$

The minimum cost cut path corresponds to the cut path of lowest capacity per unit length.

(6) **Maximum Ratio**

$$cost = -\frac{D \times L}{C}$$

The minimum cost cut path corresponds to the cut path of highest density per unit capacity per unit length.

(7) **Maximum Capacity**

$$cost = -C$$

The minimum cost cut path corresponds to the maximum capacity cut path.

(8) **Minimum Density**

$$cost = D$$

The minimum cost cut path corresponds to the cut path of minimum density.

(9) **Non-critical**

$$cost = D - C$$

The minimum cost cut path corresponds to the least "critical" cut path as defined by [Lau87].

(10) **Minimum Bottleneck**

$$cost = -\frac{C}{L}$$

The minimum cost cut path corresponds to the cut path of highest capacity per unit length.

(11) **Minimum Ratio**

$$cost = \frac{D \times L}{C}$$

The minimum cost cut path corresponds to the cut path of lowest density per unit capacity per unit length.

Various modifications to the cut path search algorithm were made to favor cut paths that approximately bisect the routing region. The motivation for these modifications comes from the observation that the cut paths found by the graph theoretic method tended to slice slivers off routing regions rather than partitioning them into roughly equal portions. While partitioning into equal portions may not be necessarily optimal, allowing highly unbalanced partitioning does little to reduce the size complexity of the routing problem. The idea behind the modifications is to exclude the extreme, sliver-producing cut paths by restricting the search for cut paths to a certain window centered around the middle of the routing region. In initial experiments, the cut path search was restricted to a user-specified window centered around the middle of the routing region. If no path was found within the window, then the region was bisected. The amount of partitioning required to produce simple, obstacle-free switchbox problems was compared for different window sizes and it was found that smaller windows did not always produce smaller amounts of partitioning. This can be attributed to the following trade-off. Bisecting a region will maximally decrease the size of the routing problems, but does not necessarily help reduce the current general area problem to the simple, target switchbox problem. On the other hand, picking a cut path based on cost functions described above will help produce simple switchbox problems, but won't necessarily help reduce the size of the routing problems much.

Thus, the bisection produces small general area problems quickly and the cost functions produce simple switchboxes slowly. A trade-off exists because small general area problems are harder than a large simple switchbox problems. At least, less is known about how to solve the general area problem than the switchbox problem. Thus, the following strategy has been implemented. Based on the cost function, search for the best cut path within a specified window around the center of the routing region. If no cut path exists within the window, then select the best cut path from outside of the window and only perform a bisection if no cut path exists. As will be seen in the next section, the choice of window size has negligible effect on solution quality, and subsequent experiments arbitrarily use the smallest sizes tested.

3.4 Results

The primary objective of the first experiments was to compare cut path selection heuristics. In an attempt to restrict data to the effects of the cut path selection heuristics only, the hierarchical decomposition was performed until the cut path selection heuristics could no longer be applied and then the number of overflows that occurred up to that point were measured. Here, an *overflow* is defined as a missing wire section or connection due to congestion in the routing region. This means that because of insufficient capacity in some area of the routing region, a portion of a net could not be implemented without shorting some other net. The stopping criteria corresponds to stopping at leaf cells that define routing problems that are simple switchboxes. The leaf cells are switchbox problems that have homogeneous routing capacity and do not contain any routing obstacles. Thus, no detailed routing was performed and the overflow values may be considered global or partition overflows. These overflows correspond to wire sections that were not assigned global routes because of insufficient capacity across a partition boundary. These values represent the actual number of missing wire sections and do not necessarily correlate with the number of nets that have missing portions. All combinations of eleven cut path cost functions, the three pin assignment cost functions, and three window sizes were tested. The cut path cost functions are *bisection, minimum capacity, maximum density, critical, maximum bottleneck, maximum ratio, maximum capacity, minimum density, non-critical, minimum bottleneck* and *minimum ratio* and are described in Section 3.3. The pin assignment cost functions are net *length*, net *projection*, and net *proximity* and are described in Chapter

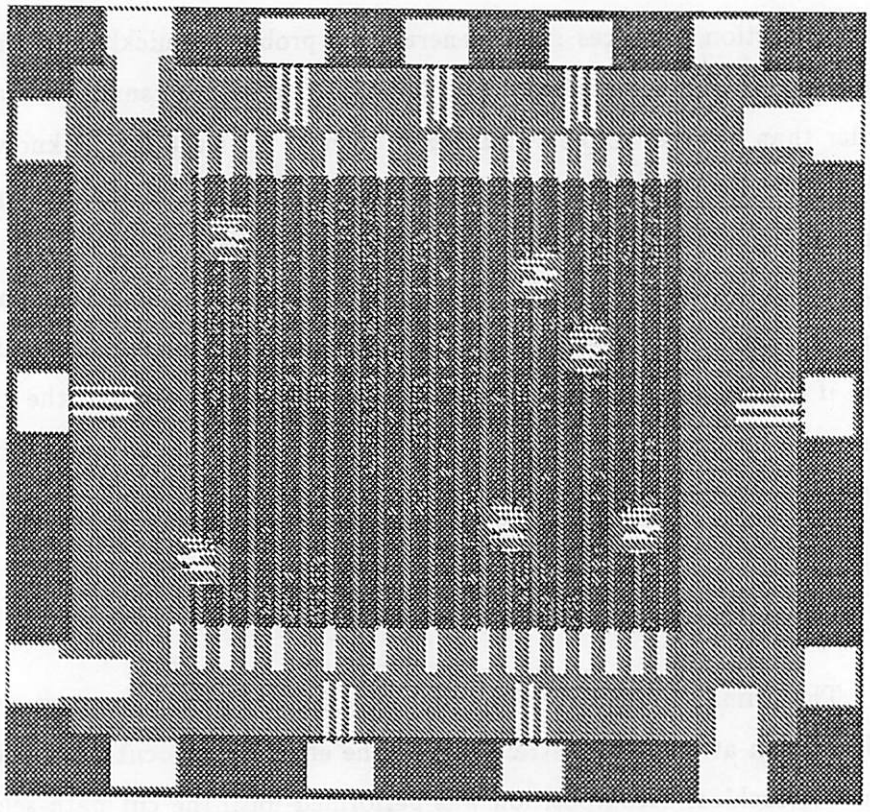


Figure 3.11: Sample routing region

4. The three window sizes are 50%, 75%, and 100%. These combinations were tried using both straight partitions and irregular partitions. All results reported are for the scan-line algorithm.

All the examples were generated by the Mariner sea-of-gates system. The router was run on the whole chip of each design. That is, the pads and the power and ground rings were included in the specification of the routing region. A sample routing region is shown in Figure 3.11. Here the routing region is shown using the geometric representation of Section 3.1. The layer denoted by cross-hatched regions is *metal 1* and the layer denoted by diagonally-hatched regions is *metal 2*. Note that the geometry of the problem is quite complex and that the problem contains large numbers of *metal 1* obstacles in small local clusters. For a hierarchical approach, this is a pathological characteristic, since the routes made at higher levels in the hierarchy assume that the capacity in the subproblems is similar to the capacity seen along the partitions. Thus, these examples are good tests of the "intelligence" of the cut paths. For these experiments, the examples consist of several

different sizes of ripple-carry adders. One adder, *adder2x8*, also includes latches on its inputs and outputs.

Figures 3.12 and 3.13 show the cost of using the different combinations of cut path cost function, window size, and pin assignment cost function for straight and irregular partitions, respectively. The cost values shown have been calculated to factor out example size while maintaining relative order. The overflow value for each combination of example, cut path cost function, pin assignment cost function, window size, and partition type was divided by the average overflow value for the corresponding example taken over all combinations of other factors. Here, partition type is either straight or irregular. The cost values shown in the figures are the average of these ratios taken over all examples. More formally, let each overflow value be specified by the quintuple (e, c, p, w, h) where e, c, p, w, and h are indices for example, cut path cost function, pin assignment cost function, window size, and partition type, respectively. Let E, C, P, W, and H be the total number of examples, cut path cost functions, pin assignment cost functions, window sizes, and partition types, respectively. Then,

$$cost(c,p,w,h) = \frac{\sum_{e=1}^{E} \frac{(e,c,p,w,h)}{avg(e)}}{E}$$

where the average overflow value for a given example, e, is

$$avg(e) = \frac{\sum_{c=1}^{C} \sum_{p=1}^{P} \sum_{w=1}^{W} \sum_{h=1}^{H} (e,c,p,w,h)}{C \times P \times W \times H}.$$

In the figures, a set of three curves is given for each cut path cost function. Within each set, a separate curve is given for each pin assignment cost function. Each curve contains three data points. From left to right, these data points correspond to window sizes of 50%, 75%, and 100%, respectively. The abscissa is labeled with mnemonics for each of the cut path cost functions as follows: **bisc** (Bisection), **minC** (Minimum Capacity), **maxD** (Maximum Density), **crit** (Critical), **maxB** (Maximum Bottleneck), **maxR** (Maximum Ratio), **maxC** (Maximum Capacity), **minD** (Minimum Density), **nonc** (Non-critical), **minB** (Minimum Bottleneck), **minR** (Minimum Ratio). The raw data for the figures are shown at the end of this chapter in Tables 3.2 to 3.4 and Tables 3.5 to 3.7 for straight and irregular partitions, respectively.

First, consider the results for straight partitions as shown in Figure 3.12. The first apparent trend is that the "critical" partitions (**minC**, **maxD**, **crit**, **maxB**, and **maxR**) perform better than the "non-critical" partitions (**maxC**, **minD**, **nonc**, **minB**,

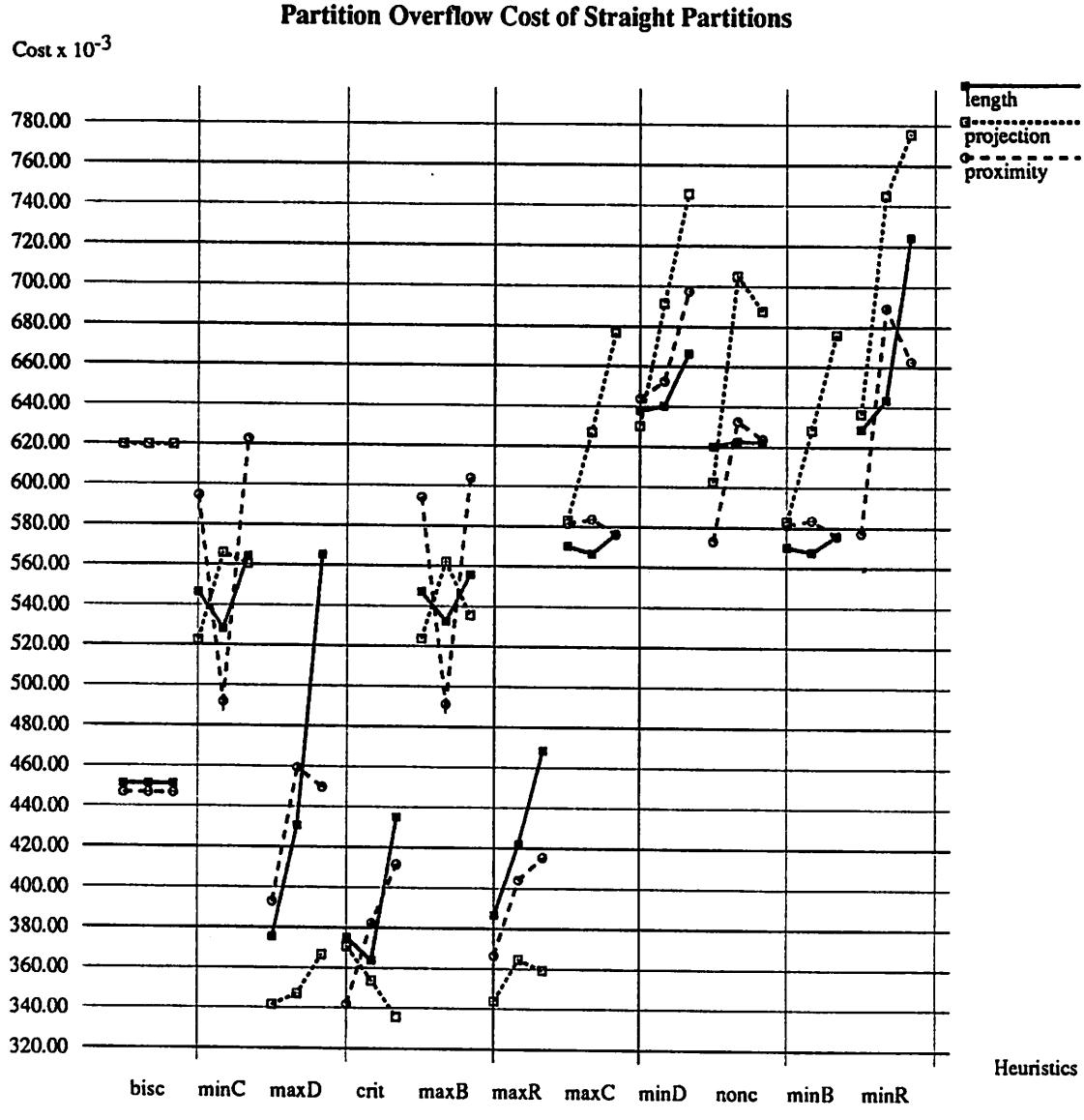


Figure 3.12: Partition overflow cost of straight partitions vs. heuristics

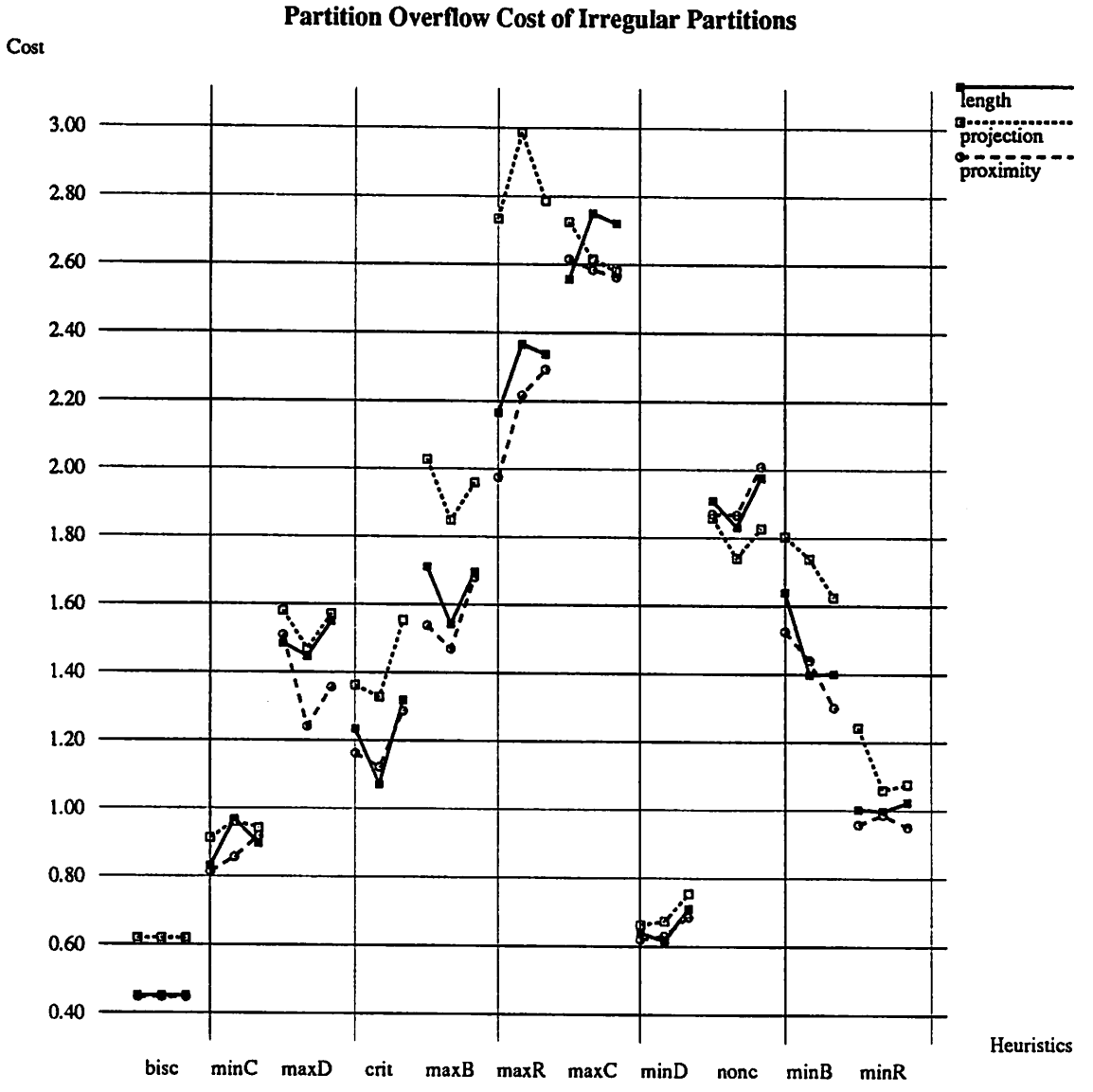


Figure 3.13: Partition overflow cost of irregular partitions vs. heuristics

and **minR**). In fact, the "non-critical" partitions always perform worse than the bisection heuristic. Among the "critical" partition heuristics, the cost functions based on maximum density outperform the cost functions based on minimum capacity. The best combinations overall are combinations of the pin assignment cost function, net *projection*, and the cut path cost functions, *maximum density*, *critical*, and *maximum ratio*. Though the nominal data indicate trends for window size for each of the best combinations, the differences within the indicated trends are small, it can be concluded that the choice of window size is relatively unimportant for this set of heuristics. These results agree in general with [Lau87].

Now, consider the results for irregular partitions shown in Figure 3.13. Unfortunately, none of the combinations of heuristics perform better than the bisection control case. After the bisection heuristic, the next best cut path cost function is *minimum density*. This result is antithesis of the result from the straight partition case where it was found that the best cut path cost functions were based on maximum density. These results are not necessarily unexpected. The sticky segment heuristics inherently cause cut paths to stick to blockages in the routing region independent of cut path cost function. Thus, the irregular cut paths are biased towards minimum capacity paths and as seen in the straight partition results, this strategy under performs the bisection heuristic. Though the capacity values are more accurate, they are usually smaller and it is not surprising that the cut paths with lower net densities have a higher success rate. Also, the irregular cut paths present harder pin assignment problems than straight partitions. The bends in the paths may represent obstacles to certain nets and are not accounted for in the linear assignment algorithm that performs the pin assignment. The details of the pin assignment are discussed more thoroughly in Chapter 4. Though these results argue against irregular cut paths, note that harder pin assignment problems and higher levels of partition overflows are acceptable if the subsequent detail routing problems are sufficiently easier and the total number of overflows in the complete routing problem is reduced.

Figures 3.14 and 3.15 show the total overflow cost of using the different combinations of heuristics for straight and irregular partitions, respectively. The data reduction and presentation is the same as in Figures 3.12 and 3.13. The difference is that the overflow values are the sum of partition and detail overflows. Here, a detail overflow is a missing wire section due to insufficient capacity in a final subproblem. Thus, the overflow values correspond to the total number of missing wire sections in the design. To perform the detail routing, the simple switchboxes of the previous experiment are hierarchically decomposed

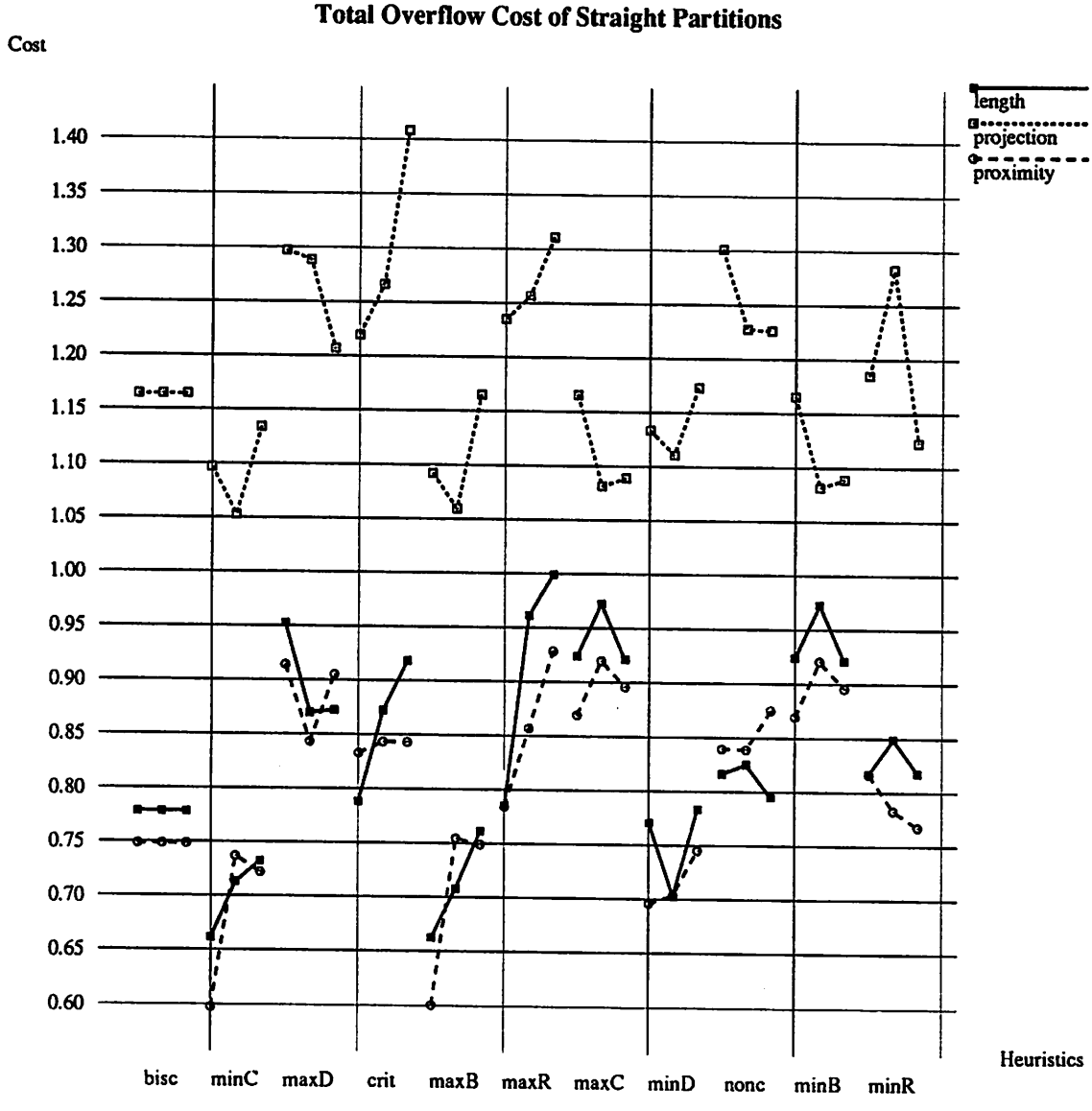


Figure 3.14: Total overflow cost of straight partitions vs. heuristics

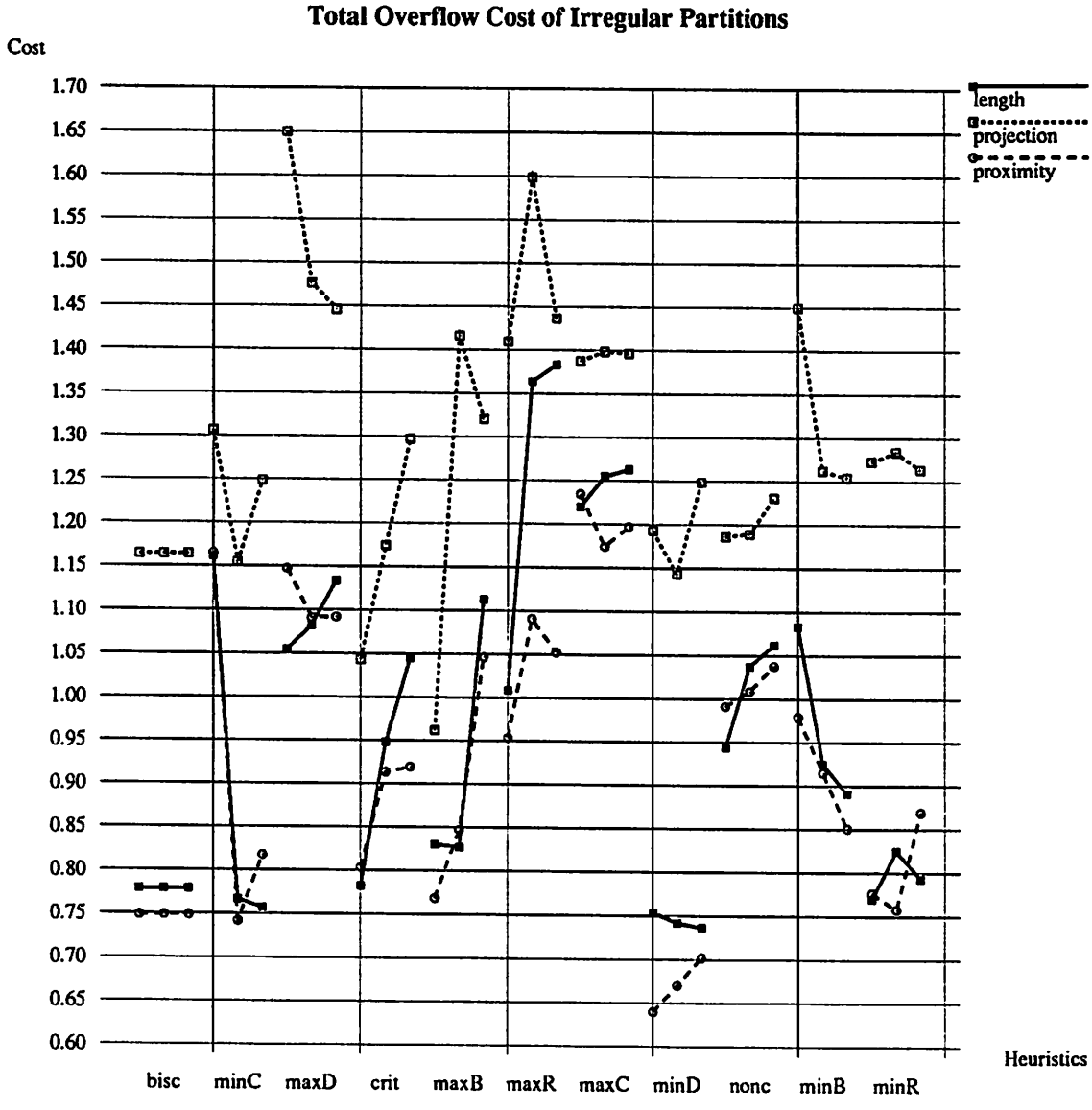


Figure 3.15: Total overflow cost of irregular partitions vs. heuristics

until the final subproblems can be routed by case analysis. Since the simple switchboxes do not contain any obstacles, the sticky segment heuristics can not be applied and the bisecting cut line that best produces subproblems with unity aspect ratios is used to partition the region. The pin assignment cost function remains the same throughout the decomposition process. The raw data for Figures 3.14 and 3.15 is shown also at the end of this chapter in Tables 3.8 to 3.10 and Tables 3.11 to 3.13, respectively.

First, consider the total overflow cost of straight partitions as shown in Figure 3.14. There are several striking features. When considering only partition overflows, the net *projection* pin assignment cost function was a member of the best combinations of heuristics. However, when considering the total number of overflows, the net *projection* cost function always performs worse than the other pin assignment cost functions independent of the other heuristics. Another feature of the graphs is that the differences between "critical" and "non-critical" partitions has disappeared. For the partition overflow case, the "non-critical" partitions always performed worse than the bisection heuristic, while for the total overflow case, the *minimum density* "non-critical" cut path cost function always performs as well or better than the bisection heuristic. Also, for the "critical" partitions, the performance of the different cost functions are the opposite of their performance when comparing only partition overflows. That is, the cost functions based on maximum density and minimum capacity performed the best and the worst, respectively, when considering only partition overflows, but when considering the total number of overflows, the cost functions based on maximum density and minimum capacity performed the worst and the best, respectively. These differences are a trade-off between ease of global routing and ease of detail routing. That is, the best global routing cost functions, while nominally producing better global routes, did so at the expense of the complexity of the detail routing problems. Another way of viewing this is that the global routing success was achieved by ignoring the real complexity of the entire routing problem. According to the graphs, the best combinations of heuristics are combinations of the pin assignment cost function, net *proximity*, and the cut path cost functions, *minimum capacity* (**minC**), *maximum bottleneck* (**maxB**), and *minimum density* (**minD**). Also, for these combinations, it appears that a window size of 50% always performs better.

Now, consider the total overflow cost of irregular partitions shown in Figure 3.15. As with straight partitions, the results are quite different than the partition overflow experiment, and furthermore, the results changed in a manner similar to the straight partitions.

The net *projection* cost function always performs worse and the cut path cost functions based on minimum capacity, *minimum capacity* (**minC**) and *maximum bottleneck* (**maxB**), and minimum density, *minimum density* (**minD**) and *minimum ratio* (**minR**), showed the most improvement. In fact, the *minimum density* cut path cost function now always performs better than the bisection heuristic. And though the *minimum density* cost function on irregular partitions does not perform better than the best straight partition heuristics, it does perform better on irregular partitions than on straight partitions. This indicates that heuristics do exist that can exploit the differences between straight and irregular partitions.

Thus, while a priori the irregular cut paths should be better than straight cut paths because they provide more accurate information, this information may not necessarily simplify the problem and/or the subsequent algorithms are not able to exploit it fully. Different or more complex heuristics and possibly a different model are needed to handle the pin assignment problem presented by the irregular cut paths. Also, the irregular cut paths require more complex geometry operations. The results show that net density is an important metric in evaluating the problem topology. For a straight cut path, this amounts to determining the planar relationship between a point and a line. For an irregular cut path, this becomes determining the intersection of a point and a polygon. Similarly, the geometry of a general area may be partitioned along a straight line by sorting the edges of the geometry and partitioning the lists appropriately, while partitioning the same area with an irregular cut path requires full-blown geometric mask operations. If these differences can be handled effectively, then the trade-off will most certainly favor irregular partitions.

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	9	23	63	76	164
	projection	9	39	102	84	267
	proximity	7	24	80	63	181
min capacity	length	5	45	90	95	198
	projection	8	39	70	78	208
	proximity	10	38	97	90	202
max density	length	7	12	66	60	159
	projection	4	15	64	64	146
	proximity	7	18	57	69	162
critical	length	4	25	66	56	165
	projection	7	20	48	54	167
	proximity	4	13	77	52	150
max bottleneck	length	5	45	90	97	195
	projection	8	39	70	80	206
	proximity	10	38	97	91	199
max ratio	length	8	14	58	71	142
	projection	5	19	57	51	152
	proximity	7	20	59	45	150
max capacity	length	9	35	86	93	220
	projection	13	27	80	95	206
	proximity	11	36	74	89	229
min density	length	12	38	93	93	240
	projection	10	38	92	97	267
	proximity	12	41	89	90	252
non-critical	length	9	37	90	124	222
	projection	8	39	86	115	232
	proximity	10	33	86	101	195
min bottleneck	length	9	35	86	93	220
	projection	13	27	80	95	206
	proximity	11	36	74	89	229
min ratio	length	12	36	77	119	219
	projection	9	42	89	104	272
	proximity	9	34	85	95	237

Table 3.2: Number of partition overflows using straight partitions - 50% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	9	23	63	76	164
	projection	9	39	102	84	267
	proximity	7	24	80	63	181
min capacity	length	5	40	88	89	211
	projection	11	39	72	79	211
	proximity	6	30	81	90	192
max density	length	6	22	70	85	163
	projection	3	23	40	74	177
	proximity	3	33	91	74	194
critical	length	3	23	70	60	159
	projection	5	19	69	45	152
	proximity	7	16	56	62	172
max bottleneck	length	5	41	88	90	211
	projection	11	35	72	86	211
	proximity	6	29	81	92	192
max ratio	length	6	25	78	64	156
	projection	6	19	41	61	188
	proximity	7	19	74	49	177
max capacity	length	8	34	80	98	247
	projection	14	33	63	103	259
	proximity	8	44	72	95	251
min density	length	11	41	93	103	229
	projection	15	48	74	102	244
	proximity	11	44	94	93	255
non-critical	length	7	51	79	126	220
	projection	10	56	97	117	247
	proximity	9	41	98	109	235
min bottleneck	length	8	34	80	99	247
	projection	14	33	63	104	259
	proximity	8	44	72	95	251
min ratio	length	10	54	70	107	238
	projection	13	57	87	108	293
	proximity	15	47	79	100	237

Table 3.3: Number of partition overflows using straight partitions - 75% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	9	23	63	76	164
	projection	9	39	102	84	267
	proximity	7	24	80	63	181
min capacity	length	6	39	103	103	189
	projection	10	40	72	86	203
	proximity	9	38	110	104	212
max density	length	11	25	87	97	208
	projection	4	22	55	68	169
	proximity	5	33	83	71	157
critical	length	6	30	67	82	141
	projection	3	16	59	62	171
	proximity	8	21	68	65	136
max bottleneck	length	6	34	102	110	188
	projection	10	34	71	81	204
	proximity	9	33	109	102	212
max ratio	length	8	29	74	68	175
	projection	4	24	51	66	159
	proximity	6	20	79	67	162
max capacity	length	8	45	97	85	187
	projection	14	52	83	98	203
	proximity	8	45	90	96	181
min density	length	9	44	101	122	244
	projection	17	51	87	110	229
	proximity	12	48	96	108	254
non-critical	length	7	45	91	121	232
	projection	10	49	95	112	271
	proximity	9	44	89	100	246
min bottleneck	length	8	45	97	85	186
	projection	14	52	83	98	201
	proximity	8	45	90	96	180
min ratio	length	10	53	106	125	258
	projection	16	54	86	105	306
	proximity	12	50	84	98	232

Table 3.4: Number of partition overflows using straight partitions - 100% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	9	23	63	76	164
	projection	9	39	102	84	267
	proximity	7	24	80	63	181
min capacity	length	9	60	104	171	343
	projection	14	59	88	215	328
	proximity	11	54	102	159	330
max density	length	19	101	278	201	570
	projection	19	85	291	283	621
	proximity	23	101	239	230	560
critical	length	26	65	136	200	520
	projection	27	61	144	256	613
	proximity	24	61	157	194	412
max bottleneck	length	29	104	233	328	576
	projection	34	102	299	428	661
	proximity	20	95	238	303	558
max ratio	length	40	147	278	383	683
	projection	45	218	346	494	798
	proximity	32	149	265	338	638
max capacity	length	37	178	350	449	972
	projection	51	165	382	449	943
	proximity	42	166	357	460	987
min density	length	12	38	83	103	249
	projection	11	32	100	118	271
	proximity	12	32	86	94	257
non-critical	length	34	113	239	334	750
	projection	33	111	241	316	724
	proximity	36	111	227	341	655
min bottleneck	length	25	115	212	267	669
	projection	30	118	237	286	729
	proximity	25	105	195	250	593
min ratio	length	16	65	134	165	406
	projection	17	87	187	211	463
	proximity	14	60	143	158	389

Table 3.5: Number of partition overflows using irregular partitions - 50% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	9	23	63	76	164
	projection	9	39	102	84	267
	proximity	7	24	80	63	181
min capacity	length	14	71	123	174	364
	projection	9	82	127	200	337
	proximity	11	66	111	148	344
max density	length	18	111	213	208	625
	projection	13	119	196	259	670
	proximity	15	75	186	211	576
critical	length	20	59	120	169	504
	projection	22	72	191	224	548
	proximity	20	51	145	201	504
max bottleneck	length	15	99	287	297	544
	projection	16	101	329	427	676
	proximity	17	88	238	306	534
max ratio	length	51	182	260	382	652
	projection	69	180	361	527	805
	proximity	53	147	240	342	664
max capacity	length	48	215	324	477	872
	projection	41	181	348	465	940
	proximity	40	197	337	465	844
min density	length	9	42	104	87	231
	projection	10	43	105	107	262
	proximity	12	44	88	87	221
non-critical	length	28	131	257	292	689
	projection	27	114	239	310	644
	proximity	31	123	247	317	701
min bottleneck	length	18	106	194	195	642
	projection	25	123	220	259	805
	proximity	22	96	205	215	608
min ratio	length	12	73	130	160	468
	projection	12	89	145	155	466
	proximity	14	69	134	141	455

Table 3.6: Number of partition overflows using irregular partitions - 75% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	9	23	63	76	164
	projection	9	39	102	84	267
	proximity	7	24	80	63	181
min capacity	length	8	85	94	173	372
	projection	9	98	82	192	352
	proximity	10	92	96	161	349
max density	length	18	128	259	212	590
	projection	18	119	204	272	696
	proximity	24	90	203	195	487
critical	length	23	111	144	186	502
	projection	16	116	284	254	550
	proximity	25	107	137	148	521
max bottleneck	length	25	103	265	314	593
	projection	24	129	318	338	768
	proximity	26	104	264	293	584
max ratio	length	52	163	302	317	712
	projection	54	181	379	497	798
	proximity	54	133	262	359	765
max capacity	length	48	195	335	476	901
	projection	41	181	350	428	950
	proximity	41	178	345	455	888
min density	length	10	52	89	106	328
	projection	11	57	123	101	275
	proximity	10	46	103	116	255
non-critical	length	30	148	276	303	743
	projection	27	130	228	311	746
	proximity	33	142	269	356	666
min bottleneck	length	20	104	170	206	652
	projection	23	124	197	207	813
	proximity	17	105	172	183	575
min ratio	length	16	66	138	184	385
	projection	14	49	183	195	482
	proximity	13	56	139	160	428

Table 3.7: Number of partition overflows using irregular partitions - 100% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	45	199	637	410	1459
	projection	70	316	772	710	2569
	proximity	51	177	592	378	1578
min capacity	length	37	210	435	360	1378
	projection	76	278	629	716	2582
	proximity	35	171	425	326	1471
max density	length	54	296	625	528	1815
	projection	74	386	863	745	2868
	proximity	50	245	595	620	1880
critical	length	50	215	548	432	1506
	projection	74	319	846	733	2775
	proximity	42	216	665	502	1750
max bottleneck	length	37	210	435	363	1375
	projection	76	278	629	705	2584
	proximity	35	171	425	333	1470
max ratio	length	43	237	565	420	1787
	projection	71	326	1002	637	3032
	proximity	42	243	560	417	1913
max capacity	length	59	238	598	577	1824
	projection	72	304	732	761	2926
	proximity	51	232	653	482	1832
min density	length	43	221	552	435	1723
	projection	73	261	788	726	2864
	proximity	41	184	530	381	1831
non-critical	length	46	227	558	499	1666
	projection	110	312	788	675	2728
	proximity	62	206	571	448	1690
min bottleneck	length	59	238	598	577	1824
	projection	72	304	732	761	2926
	proximity	51	232	653	482	1832
min ratio	length	50	233	548	459	1775
	projection	69	312	802	752	3294
	proximity	56	186	566	499	2056

Table 3.8: Number of total overflows using straight partitions - 50% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	45	199	637	410	1459
	projection	70	316	772	710	2569
	proximity	51	177	592	378	1578
min capacity	length	48	215	463	347	1483
	projection	66	288	624	681	2591
	proximity	52	229	469	330	1408
max density	length	42	251	613	556	1681
	projection	69	377	888	767	2893
	proximity	43	196	581	642	1802
critical	length	55	249	616	447	1559
	projection	90	326	826	704	3012
	proximity	51	232	629	441	1839
max bottleneck	length	48	214	463	338	1483
	projection	66	294	624	683	2591
	proximity	52	240	469	346	1408
max ratio	length	66	268	659	477	1843
	projection	83	355	786	710	2955
	proximity	39	248	683	494	1785
max capacity	length	66	273	621	526	1823
	projection	61	263	777	714	2854
	proximity	61	242	620	519	1698
min density	length	39	172	549	427	1703
	projection	71	301	731	646	2905
	proximity	43	192	501	390	1661
non-critical	length	54	197	589	490	1670
	projection	105	275	821	600	2643
	proximity	68	205	559	406	1711
min bottleneck	length	66	273	621	527	1823
	projection	61	263	777	715	2854
	proximity	61	242	620	521	1698
min ratio	length	68	180	553	495	1743
	projection	94	302	758	827	3033
	proximity	54	177	574	450	1702

Table 3.9: Number of total overflows using straight partitions - 75% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	45	199	637	410	1459
	projection	70	316	772	710	2569
	proximity	51	177	592	378	1578
min capacity	length	37	249	553	330	1204
	projection	65	309	838	639	2389
	proximity	46	197	562	339	1306
max density	length	44	224	679	547	1688
	projection	70	310	851	754	2956
	proximity	51	223	665	578	1608
critical	length	69	249	611	435	1485
	projection	103	390	880	726	2802
	proximity	39	247	692	455	1471
max bottleneck	length	37	257	552	381	1201
	projection	65	330	828	672	2385
	proximity	46	224	561	340	1296
max ratio	length	71	256	709	505	1796
	projection	92	329	911	708	2957
	proximity	76	212	642	456	1742
max capacity	length	66	219	632	521	1619
	projection	61	245	851	707	2471
	proximity	61	204	637	536	1583
min density	length	55	168	595	444	1487
	projection	81	275	873	638	3125
	proximity	51	183	576	370	1566
non-critical	length	54	176	643	418	1601
	projection	105	278	805	605	2761
	proximity	68	208	607	442	1584
min bottleneck	length	66	219	632	521	1619
	projection	61	245	851	707	2469
	proximity	61	204	637	536	1582
min ratio	length	64	173	604	425	1698
	projection	69	239	881	703	3258
	proximity	58	163	592	394	1666

Table 3.10: Number of total overflows using straight partitions - 100% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	45	199	637	410	1459
	projection	70	316	772	710	2569
	proximity	51	177	592	378	1578
min capacity	length	126	185	661	612	1571
	projection	128	195	779	813	2248
	proximity	150	152	631	512	1482
max density	length	57	299	666	702	2387
	projection	89	342	1354	1112	3449
	proximity	59	306	844	726	2560
critical	length	45	145	557	621	1636
	projection	55	182	595	1034	2197
	proximity	46	121	575	703	1620
max bottleneck	length	71	164	433	571	1210
	projection	55	187	600	824	1682
	proximity	58	165	416	553	1170
max ratio	length	83	261	547	556	1210
	projection	101	393	677	925	1535
	proximity	61	327	490	518	1155
max capacity	length	77	321	657	875	2295
	projection	87	362	778	981	2902
	proximity	79	336	690	826	2444
min density	length	38	219	575	422	1483
	projection	77	269	943	672	2646
	proximity	29	174	566	345	1356
non-critical	length	50	238	613	691	2044
	projection	73	328	676	797	2879
	proximity	53	262	657	682	1916
min bottleneck	length	75	288	712	592	1962
	projection	100	362	950	854	2878
	proximity	53	283	682	580	1845
min ratio	length	40	246	550	402	1730
	projection	77	281	853	918	2669
	proximity	48	228	561	380	1550

Table 3.11: Number of total overflows using irregular partitions - 50% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	45	199	637	410	1459
	projection	70	316	772	710	2569
	proximity	51	177	592	378	1578
min capacity	length	42	158	552	584	1560
	projection	77	186	827	882	2088
	proximity	49	160	520	486	1466
max density	length	65	330	683	607	2268
	projection	76	462	804	1030	3335
	proximity	73	304	621	675	2219
critical	length	81	192	594	550	1013
	projection	101	190	574	937	1680
	proximity	84	162	529	571	1082
max bottleneck	length	62	167	497	579	1022
	projection	113	203	1134	866	1441
	proximity	65	167	505	592	1011
max ratio	length	162	296	454	703	1128
	projection	123	395	713	1145	1600
	proximity	86	362	356	662	1146
max capacity	length	89	314	678	849	2107
	projection	93	341	742	1036	2976
	proximity	80	302	636	803	2084
min density	length	45	188	547	431	1529
	projection	80	249	841	665	2572
	proximity	38	162	517	410	1432
non-critical	length	75	257	568	688	1744
	projection	82	287	719	781	2734
	proximity	77	222	592	662	1811
min bottleneck	length	56	290	601	481	1915
	projection	78	346	771	804	2978
	proximity	55	269	613	504	1903
min ratio	length	57	221	559	433	1473
	projection	83	316	939	729	2759
	proximity	47	187	547	454	1540

Table 3.12: Number of total overflows using irregular partitions - 75% window

Cost Functions		Examples				
Cut Path	Pin Assgt	adder2	adder4	adder8	adder2x8	adder16
bisection	length	45	199	637	410	1459
	projection	70	316	772	710	2569
	proximity	51	177	592	378	1578
min capacity	length	33	167	558	614	1433
	projection	97	275	685	849	1997
	proximity	60	170	597	474	1420
max density	length	65	330	757	662	1900
	projection	80	480	693	981	3350
	proximity	79	304	623	621	2092
critical	length	95	276	452	589	1081
	projection	86	286	898	842	1570
	proximity	97	222	364	477	1091
max bottleneck	length	95	238	560	741	1318
	projection	86	311	833	898	1682
	proximity	84	229	610	657	1275
max ratio	length	163	374	540	476	1225
	projection	115	406	610	894	1842
	proximity	87	246	514	692	1252
max capacity	length	89	308	672	888	2179
	projection	92	370	827	892	2871
	proximity	83	320	636	788	2117
min density	length	46	176	499	481	1709
	projection	85	290	895	728	3232
	proximity	42	164	578	390	1642
non-critical	length	77	262	551	733	1935
	projection	86	330	658	798	2652
	proximity	79	245	576	674	1820
min bottleneck	length	52	245	619	524	1837
	projection	79	372	862	631	2922
	proximity	48	233	636	477	1791
min ratio	length	43	234	562	449	1671
	projection	69	324	893	829	2566
	proximity	68	223	559	436	1838

Table 3.13: Number of total overflows using irregular partitions - 100% window

Chapter 4

Net Decomposition

Net partitioning requires determining which nets must cross which subregion boundaries and for each net that crosses a partition, determining how many times it crosses and where it should cross. Consider the example in Figure 4.1a. Here an example routing region is shown with two of its nets and a proposed partition. The nets are shown by dashed lines and the partition is shown by a dotted line. Given this partition, it must be decided which nets belong to which subproblems. In this case, net B is contained fully in the lower subregion, while net A is contained in both the upper and lower subregions. Thus, net A will be assigned at least one net crossing position along the partition and net B will not be assigned any crossing positions. In more complex strategies, net B also could be made to cross the partition, but for these experiments, this possibility is ignored. Given that net A must cross the partition, the number times that net A will be allowed to cross the partition must also be determined. Figure 4.1(a) shows an example of assigning net A one crossing position and Figure 4.1(b) shows an example of assigning net A two crossing positions.

4.1 Net Partitioning

For the hierarchical decomposition described in this report, determining which nets cross which boundaries requires few heuristics. Since the subdivision of routing regions is a two-partition, any net that has at least one pin on both sides of the partition must cross the partition. If a net has a pin or pins that lie exactly on the partition and the remaining pins all on one side of the partition, then the net is not partitioned and is added to the subproblem corresponding to the subregion containing the non-partition pins. In all other

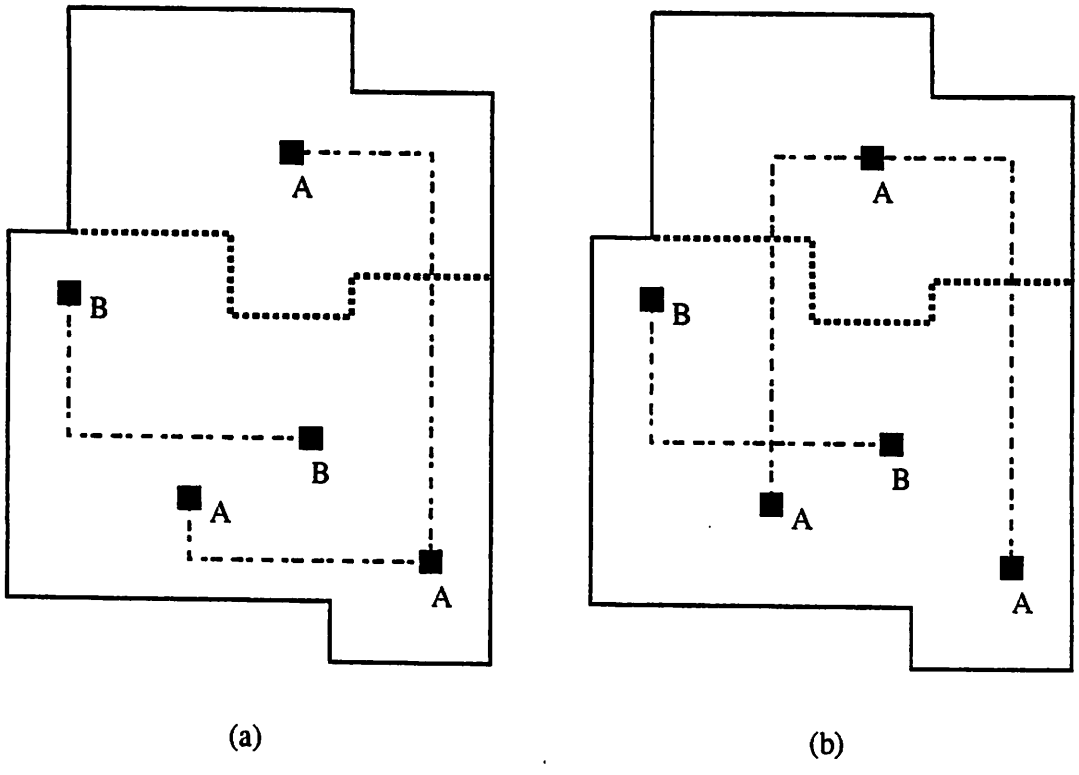


Figure 4.1: Net Decomposition

cases where a net has pins that lie on the current partition, the net is partitioned and its partition pins are distributed arbitrarily between the two subproblems. If a routing grid is employed, then this is not a problem because all cut paths are constrained to fall between routing grids. However, in a grid-less paradigm these situations must be considered. In the current implementation, both gridded and grid-less approaches have been coded, though only the gridded method has been fully tested. All the examples in this report were run using an explicit routing grid. Unfortunately, determining the location of a point (or pin position) relative to an arbitrary rectilinear path is not trivial. In general, the intersection of the point with each of the corresponding subregions must be calculated. Intersection with both of the subregions means that the pin lies on the cut path. For the case of pseudo-pins, intersection calculations can be avoided by storing previously calculated pin-partition and partition-region topological relationships in the data structures. With this information, calculating the intersection of a pseudo-pin with a subregion transforms into two membership queries.

For each net that crosses the current partition, the number of times that the net crosses the partition must also be decided. The simplest course of action is to only allow each net to cross once. However, this may not be optimal. Allowing only one crossing forces any remaining connections to be made within the subproblems which are smaller and more likely congested. Allowing multiple crossings would make more effective use of capacity at the higher levels of the hierarchy and hopefully would avoid adding to the local congestion of the subproblems. The minimum spanning tree of a net can be used to determine the number of times that a net will be allowed to cross a partition. In particular, the number of crossings can be set equal to the number of edges of the minimum spanning tree of the net that cross the partition. Deletion of these crossing edges represents a partitioning of the net. The remaining connected components correspond to subnets of the original net and can be assigned to the appropriate subproblems. Some results on the effect of using this means of allowing multiple crossings are shown in Table 4.1. The influence of multiple crossings was checked on the following irregular cut path cost function combinations. The combinations are the *minimum capacity* cut path cost function with the net *proximity* pin assignment cost function and a window size of 75%, and the *minimum density* cut path cost function with the net *proximity* pin assignment cost function and a window size of 50%. Since, irregular cut path cost functions based on combinations of minimum density and minimum capacity performed the best, this choice of combinations allows examination of the effects

Examples	Irregular Cut Path Cost Functions			
	Minimum Capacity		Minimum Density	
	Multiple Crossings	Single Crossing	Multiple Crossings	Single Crossing
adder2	51	49	34	29
adder4	174	160	175	174
adder8	548	520	569	566
adder2x8	577	486	362	345

Table 4.1: Multiple versus single net crossings on irregular cut paths

of both characteristics. As seen in the table, using multiple crossings always performs worse than using single net crossings. This is consistent with the result that minimum density irregular cut paths perform better since allowing multiple crossings increases the net density of subsequent subproblems.

4.2 Pin Assignment

After partitioning the nets into subnets and determining the number of pseudo-pins that will be placed on the partition, pseudo-pins must be assigned to particular locations on the partition. In this work, pseudo-pins are allowed to float anywhere along their partition subject to future partitioning constraints. That is, the pseudo-pins may move anywhere along the partition as long as such movement does not change their membership in *any* subregion of the hierarchy. However, exact locations are assigned to the pseudo-pins because their positions influence the choice of subsequent partitions. This assignment may be thought of as an initial best guess.

For the case of two-partition hierarchical decomposition, there is a natural formulation of the problem of assigning pins to partition positions. This pin assignment problem is defined as follows.

Given

i the ith pin to be assigned $i = 1, 2, \ldots, n$
j the jth position on the partition $j = 1, 2, \ldots, m$
cap_j the number of pins which may be assigned to position j
$cost_{ij}$ the cost of assigning pin i to position j

determine a minimal cost assignment of pins to positions such that all pins are assigned and no position has its capacity exceeded.

This a linear assignment problem that is defined by the following linear program. Let

$$x_{ij} = \begin{cases} 1 & \text{if pin } i \text{ is assigned to position } j \\ 0 & \text{otherwise.} \end{cases}$$

Solve

$$\text{minimize} \sum_{i,j} cost_{ij} x_{ij}$$

subject to

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} &\leq cap_j \quad \forall j \\ \sum_{j=1}^{m} x_{ij} &\leq 1 \quad \forall i \\ x_{ij} &\geq 0 \quad \forall i,j \end{aligned}$$

A package has been written that solves this problem by formulating it as a minimum cost flow problem [Law76]. The time complexity of the algorithm is $O(n^3m + m^2n^2)$ where n is the number of pins that need assignment to a position and m is the number of positions.

In the program, this problem is actually solved in two phases. First, pins are assigned to segments of the cut path. These segments are the segments that arise naturally from the bends in the path and from changes in capacity along the path. That is, each segment is a straight line segment and is homogeneous in the specific layers that may be used to route across it. The second and final phase is to take the pins assigned to a particular segment and assign them to a particular track on this segment.

Now, to actually perform the pin assignment, heuristics must be applied to determine the capacity vector, cap_j, and the cost matrix, $cost_{ij}$. These heuristics are very important because they embody most of the decision making that occurs at each level of the hierarchy. The pin assignment constrains the topology of the subsequent problems directly through the selected pin positions and indirectly through its influence on subsequent partition selection.

There are several ways to calculate the capacity of a partition. The most straightforward way is to follow the typical wiring model conventions and assume that certain routing layers have associated legal orientations. For example, all vertical wire segments may be required to use *metal 1* and all horizontal wire segments may be required to use *metal 2*. Thus, for each segment of the cut path a capacity could be calculated with respect to the

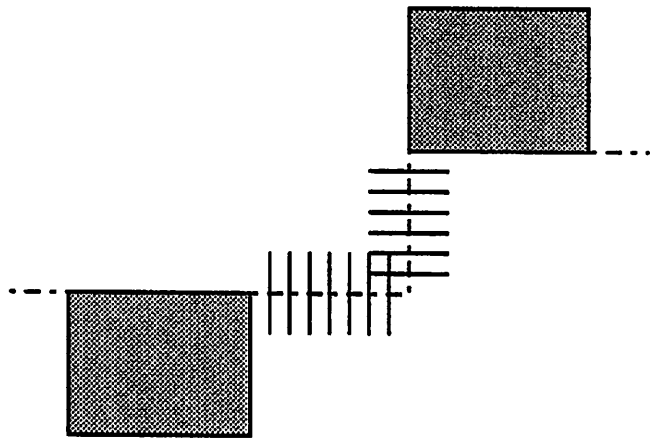


Figure 4.2: Cut path bend capacity

layers associated with the direction perpendicular to the segment. Unfortunately, there are times when enforcing standard wiring model constraints are too restrictive. Routers such as [RSVS85] and [SSV87] have shown that it can be advantageous to allow routes to selectively break the wiring model and in the ideal case, more efficient routes can be achieved by relaxing the wiring model entirely. For example, wrong-layer wire segments are needed to cross the power and ground rings that surround the chip. It is also easy to think of situations where wrong-layer dog-legs or wrong-layer wires over cells will be needed to complete a route. However, arbitrarily allowing nets to break the wiring model is probably not a good idea. In particular, it will create a problem when computing the capacity of bends in a cut path. Consider the example in Figure 4.2. The lines crossing the cut path represent assigned net crossings. If the wiring model is not followed, then the capacity on the concave side of the bend is much less than would be calculated by summing the capacities of the adjacent segments of the corner. One alternative is to use a more restrictive approximation and assume zero capacity for all segments that are perpendicular to the overall orientation of the cut path. Unfortunately, pathological cases exist where this method will be much too conservative. For example, consider the case where the perpendicular segment of the bend in the cut path represents most of the capacity of the bend. Assigning this perpendicular segment zero capacity will drastically underestimate the capacity of the cut path.

Fortunately, modifying the capacity model is not the only way that deviations from the wiring model can be influenced. With the pseudo-pin assignment model, the capacity calculations can be performed, assuming that wires will break the wiring model if

Examples	Constant Penalty Factor		Dynamic Penalty Factor	
	C	D	C	D
adder2	1.20	0.81	1.20	0.81
adder4	1.30	1.27	1.33	1.29
adder8	1.80	1.54	1.88	1.47
adder2x8	1.49	1.11	1.65	1.11
Notes: C = minimum capacity cost function D = minimum density cost function				

Table 4.2: Overflow ratios for penalty factors versus the standard wiring model

the standard routing layer is blocked and then entries in the cost matrix can be modified to penalize wrong-layer wires. In these experiments, two styles of penalties have been tried. The first variation is to simply multiply the cost of a wrong-layer wire by a constant factor. The default value of this factor in the current experiments is two. The second option is to calculate a factor dynamically. The constant-factor method allows us to break the wiring model, but does not give us much control over when this occurs. In particular, it is desirable to allow wiring model deviations at the higher levels of the hierarchy where they may be needed, but not allow them at the lower levels where the subproblems are not large enough to deal with the wrong-layer segments. The dynamic penalty factor is an attempt to change the layer preference constraints as the router travels down the hierarchy. The penalty factor is calculated as

$$\text{factor} = 1 + \frac{\text{length of original root problem}}{\text{length of current subproblem problem}}.$$

If the cut path orientation is vertical, then *length* is equal to the height of the problem and if the the cut path orientation is horizontal, then *length* is equal to the width of the problem. Table 4.2 compares enforcing the standard wiring model with the two types of penalty factors. Each value in the table is the ratio of the number of total overflows that results from using the given type of penalty factor to the number of total overflows that results from enforcing the standard wiring model. Data was obtained for both the best *minimum capacity* and the best *minimum density* cost combinations. Only one example, *adder2*, was ever improved by using the penalty factors. This confirms that situations exist where breaking the wiring model is advantageous, Unfortunately, it also shows that, in general, these particular instances of using penalty factors to allow deviations from the wiring model are not better than strictly enforcing the wiring model. That is, too many

wires are allowed to deviate from the wiring model and the increased number of overflows can be attributed to increased problem complexity caused by the larger number of wrong-layer wires. While increasing the value of the penalty factors may improve the results, it is not clear that any particular value will be best for all examples. It seems that these metrics oversimplify the problem of deciding when to break the wiring model. Given that underlying detail routing algorithm generally trys to follow the standard wiring model, the decision to break the model must be made quite selectively and preferably in situations where it is known that this disruption will not adversely affect neighboring routes. More work is required to understand how to model the effect of wrong-layer global wires.

The primary cost of assigning a particular net to cross a particular position of the partition also must be determined. So far three simple cost functions have been tested. These cost functions are referred to as net *length*, net *projection*, and net *proximity* in the tables and figures.

The net *length* cost function is a simple Manhattan net length metric. Assume that the decomposition of a given routing region results in the two subregions A and B. Define P_i^A and P_i^B as the sets of pins of net i that are contained in subregions A and B, respectively. Let the j^{th} position on the partition be at the point (x_j, y_j) and the position of a pin, p, be at the point (x_p, y_p). Then, the cost of assigning the pseudo-pin for net i to the j^{th} position on the partition is

$$\sum_{p \in P_i^A, P_i^B} (\mid x_j - x_p \mid + \mid y_j - y_p \mid).$$

The net *projection* cost function is similar to the cost function suggested by [Lau87]. The cost of assigning the pseudo-pin for net i to the j^{th} position on the partition is the sum of two components — one for each of P_i^A and P_i^B. Thus, an attempt is made to minimize the cost relative to both subregions. Consider the component corresponding to P_i^A. Project the bounding box of the pins in P_i^A onto the partition. If the j^{th} position falls within this projection, then the $cost(P_i^A) = 0$. If the j^{th} position falls outside this projection then $cost(P_i^A) = 1000\times \mid m \mid$ where m is the slope of the line passing through (x_j, y_j) and the nearest corner of the bounding box.

The net *proximity* cost function is similar to the net *length* cost function, except that it only considers the two pins, one from P_i^A and one from P_i^B that are closest to (x_j, y_j).

Thus, the cost of assigning the pseudo-pin for net i to the j^{th} position on the partition is

$$\min_{p \in P_i^A} (\mid x_j - x_p \mid + \mid y_j - y_p \mid) + \min_{p \in P_i^B} (\mid x_j - x_p \mid + \mid y_j - y_p \mid).$$

Recall from Chapter 3 that when using straight cut paths and considering only partition overflows, the net *projection* cost function is the best. In terms of the total number of overflows, the best combinations of heuristics for both straight and irregular cut paths used the net *proximity* pin assignment cost function. This result agrees with the view of the partitions as accurately representing capacity only in a narrow region surrounding the cut paths. Given this assumption, a cost function that concentrates on pin assignments relative to the closest pins is expected to perform better than cost functions that apply the capacity information to far away pins. This is particularly important for irregular cut paths where the bends in the path represent "unseen" obstacles to pins far away from the partition. The net *proximity* cost function tries to assign pseudo-pins to positions that will be optimal for the closest pins of the subnets. This assumes that the final route will and should be between the pin of each subnet that is closest to the pseudo-pin position. Under this model, it is unlikely that a pseudo-pin will be assigned to a position such that one of these short direct routes will have to cross a bend.

More complex and more intelligent cost functions could be used. For example, the ease of routing from a net terminal to a particular partition position might be estimated by estimating the routing capacity and/or the number of obstacles on a Manhattan path between the terminal and the partition position. Also, the capacity of a segment might be reduced, depending on the number of obstacles that block a straight approach to it. Unfortunately, for general routing areas, all of these cost functions require geometric intersection calculations. Since these calculations are very complex, experiments with these cost functions have been postponed pending a more tailored implementation of the data structures. More importantly, the cost functions should also account for potential interactions between nets. That is, the current algorithm minimizes total cost relative to individual net costs that are calculated with respect to the individual net. A better, though maybe impossible, method would be to include net interactions more directly in the individual net cost values. Thus, minimizing total cost would correspond more closely to minimizing global cost instead of individual costs.

Chapter 5

Subproblem Routing

The problem of routing the final subproblems and the problem of determining when to stop partitioning are closely coupled. In the following, two methods of performing the final detail routing and the corresponding choices of stopping criteria are discussed. The two detail routing experiments are pattern routing and maze routing. The related issue of subproblem routing order will also be discussed in this chapter.

5.1 Pattern Routing

Two-layer pattern routing code has been implemented to perform the final subproblem routing, though extending this pattern router to three or more layers is straightforward. This is a variation on Burstein and Pelavin's [BP83b] approach of enumerating all possible routes for a 2×2 grid. In this work, stopping criteria were chosen so that a reasonable case analysis of the final subproblems would be possible. Currently, the pattern router considers 151 basic cases determined by number of nets and relative pin topology. Routing layer choices are made within each case. Thus, the most obvious extension to n routing layers will not increase the number of basic cases, but will increase the complexity of each case. In such an implementation, a layer assignment algorithm is expected to narrow the choice of routing layers for each subproblem to a 2- or 3-layer subset of the possible layers. Thus, the analysis for each case only needs extension to allow for the possibility of three routing layers. In an ideal system, this layer assignment will be the natural result of a three-dimensional hierarchical decomposition rather than a separate analysis.

The stopping criteria consist of the following rules.

1. The routing region must be a simple switchbox.
   This means that the routing region is defined by identical rectangles on all routing layers specified for the subproblem.

2. The routing region is free of obstacles.
   This guarantees homogeneous routing capacity across the region.

3. The number of pins on each side of the switchbox is less than or equal to one.
   This restricts subproblems to having either zero, one, or two nets.

4. The number of pins internal to the subproblem is less than or equal to one.
   This is a manifestation of the partitioning model.

The last constraint is an unfortunate consequence of dealing with internal pins. Given a problem with internal pins, the partitioning process must not be allowed to cut across any of these pins, otherwise it will be possible to create a subproblem with a pin at a corner of its routing region. This situation is impossible to route. To guarantee that partitions will not cut through an internal pin, a partitioning model must be enforced that will never produce a cut path segment that intersects an internal pin. For gridded approaches, this amounts to always partitioning on the dual graph or intergrid lines of the specified routing grid-graph. For grid-less paradigms, the partitioning must occur on the dual graph of the virtual routing grid induced by pin positions. Therefore, partitions are made around the internal pins and internal pins are present in the final subproblems. This inconsistency in the pin model also greatly increases the amount of case analysis that must be performed to implement the detailed routing. Much enumeration can be avoided by using an improved model that will be discussed later.

Alternately, if either of the following conditions is met, the partitioning will stop.

1. The routing region does not contain any nets.
   This is the degenerate case.

2. The region is too small to partition.
   A switchbox with height and width equal to less than two of the appropriate routing pitches can not be partitioned.

From these constraints, all the different types of subproblems that may occur can be enumerated. First consider when a subproblem contains only one net and for the moment,

assume that the subproblem does not contain any internal pins. Under these conditions, all possible 2-, 3-, and 4-pin nets can be enumerated as in Figure 5.1. This class of subproblem can always be routed, even if the subproblem has only one routing layer specified. With respect to mirroring and rotation operations, the eleven cases in Figure 5.1 can be reduced to four. The four cases are a two-terminal connection between adjacent sides (*L-net*), a two-terminal connection between opposite sides (*I-net*), any three-terminal connection (*T-net*), and any four-terminal connection (*X-net*). A representative example of each of these four cases is marked in Figure 5.1 by background shading. Also, the abstraction only depends on the pin-edge assignments and is independent of pin position and switchbox size. Thus, in general, the pins do not have to align and the routing connections may contain jogs instead of being straight line segments. Note that this generalization does not change the number of classifications.

If a single-net problem includes an internal pin then the situation becomes more complicated. In general, an internal pin may line-up with an edge pin or be to one side or the other of an edge pin. The possible relationships are shown in Figure 5.2. In this figure, the four basic single-net connections are illustrated and *X*'s depict possible internal pin positions. The internal pin positions are shown after reduction under mirroring and rotation. Thus, for a subproblem with a single net containing a single internal pin, there are six *L-net* cases, two *I-net* cases, six *T-net* cases, and three *X-net* cases. Note that it is possible for a single net to contain only a single edge pin and a single internal pin. These cases may be handled as an *L-net* or *I-net* of Figure 5.1 by pretending that the internal pin is an edge pin. Similarly, some of the *L-net* and *I-net* cases of Figure 5.2 may be treated as *T-net* cases, and certain *T-net* cases can be processed as *X-net* cases. There are four cases of single nets without internal pins and seventeen cases of single nets with internal pins for a total of twenty-one single-net cases.

Now, consider a subproblem that contains two nets. If the subproblem does not contain an internal pin, then the different possible situations may be enumerated as shown in Figure 5.3 for the two nets, A and B. Note that the three cases shown are actually only two cases. These two cases are the two two-pin cross connections (*2I-net*) and the two two-pin adjacent connections (*2L-net*). Unlike the single-net case, the ability to route these cases depends on the number of available routing layers, the size of the switchbox, the legal range of pin positions, and the legal layers of each pin. Fortunately, after an analysis of these factors, double-net problems often can be converted into canonical forms

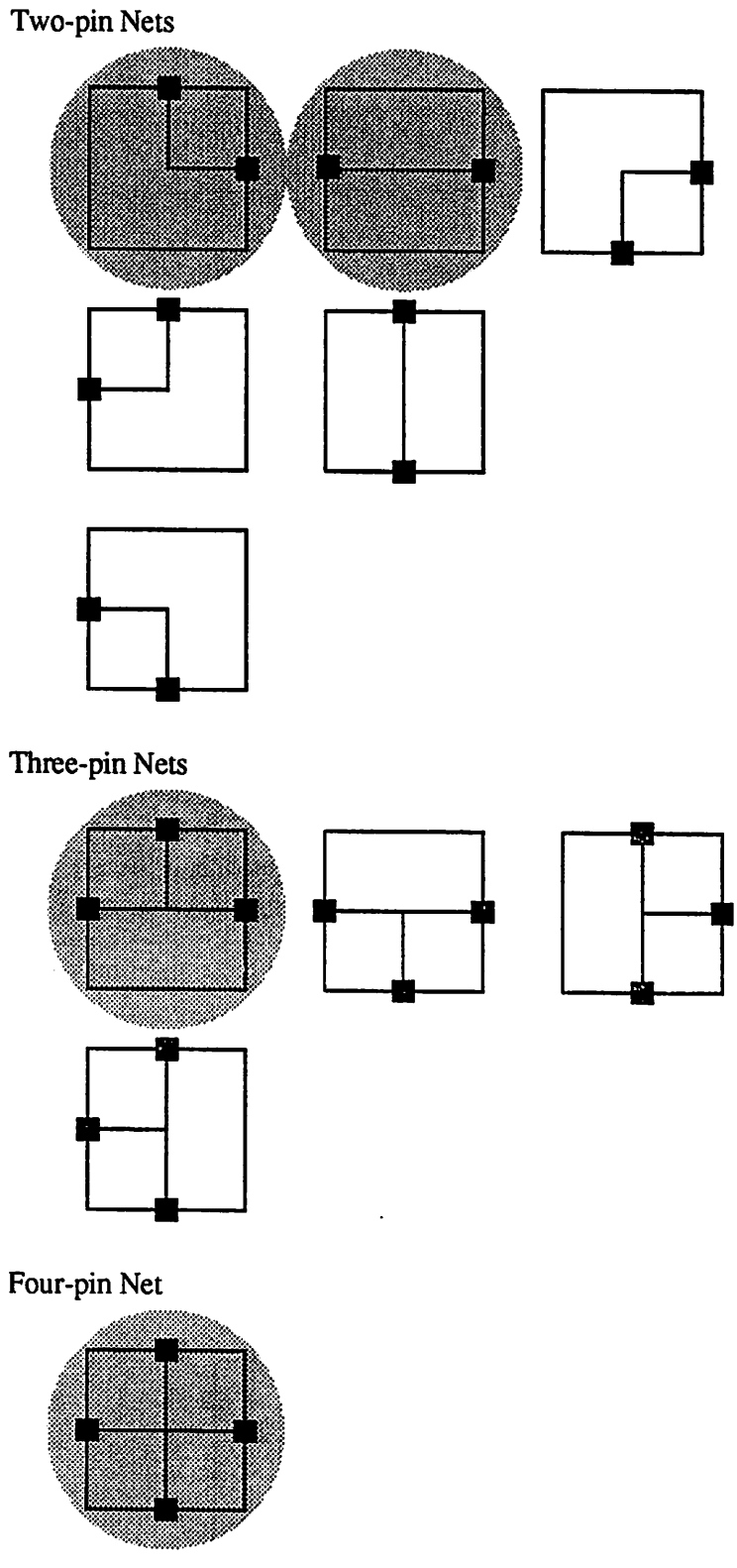


Figure 5.1: Single-net configurations without internal pins

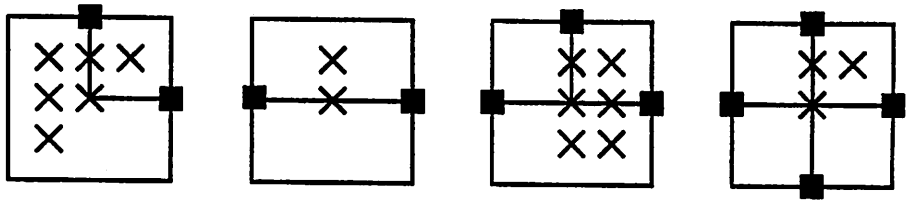


Figure 5.2: Single-net configurations with internal pins

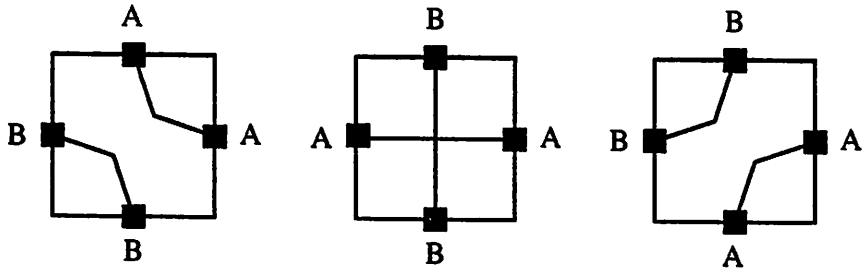


Figure 5.3: Double-net configurations without internal pins

that use combinations of single-net patterns. In general, an attempt is made to formulate each double-net case as an abstract 2 × 2 grid problem. If all sides of the switchbox are greater than or equal to two routing pitches, then this is always possible. Figure 5.4 shows an example 2 × 2 grid abstraction for the *2I-net* case. In this figure, there are four grid cells and each grid may correspond to one or more actual routing tracks. The lines delimit the boundaries of the grid cells and the small black boxes are pseudo-pins. The edge of a grid cell that is crossed by a net is denoted by each pseudo-pin. The direction of a net segment may not change on grid cell boundary. All direction changes must be made within the grid cells. In an actual 2 × 2 grid problem, if two nets enter the same grid cell, they must be on separate routing layers. In the general case, this is a conservative restriction

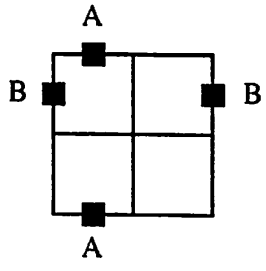


Figure 5.4: Example 2 × 2 grid abstraction

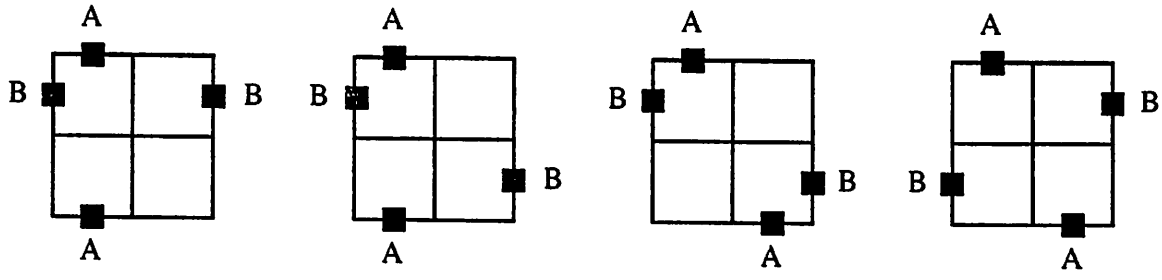


Figure 5.5: *2I-net* pin-position configurations without internal pins

since the size of an abstract grid cell may be larger than one routing pitch. The pseudo-pins of the two nets define the entry positions of the nets along the edges of the grid cell and actually, the pseudo-pins of two nets that enter the same grid cell must be assigned separate layers only if they are within a minimum routing distance of each other. Here, minimum routing distance means that both the x-coordinates and the y-coordinates of the pins differ by only one routing pitch. However, if the layer specifications of the pins satisfy the layer constraints of a 2×2 grid problem and the pins are not all restricted to the same layer, then the problem always can be routed. Thus, in practice, the router always trys to form feasible 2×2 grid abstractions. This may involve judicious choice of pin-to-grid assignments or movement of floating pseudo-pins. Note, that pins on the same grid do not need to be physically aligned. If a problem is too small to be formulated as a 2×2 grid, then under certain conditions it still may be routed. These special cases include partitioning the nets into different layers and forming physically disjoint subproblems, and were not included in the count of basic cases.

First, consider the *2I-net* case. Suppose that the problem can be abstracted to a 2×2 grid problem. Then, after reduction under mirroring and rotation, the problem will have one of the four possible pin topologies shown in Figure 5.5. If the the problem can not be abstracted to a 2×2 grid problem because the length of a side of the switchbox is less than two routing pitches, then the problem still may be routed if the pin layer-specifications allow the partitioning of the two nets onto different layers. In this case, each net will be routed as a single-net problem confined to a specific routing layer.

Now, consider the *2L-net* case. If the problem can be abstracted to a 2×2 grid problem, then after reduction under mirroring and rotation, it will have one of the six possible pin topologies shown in Figure 5.6.

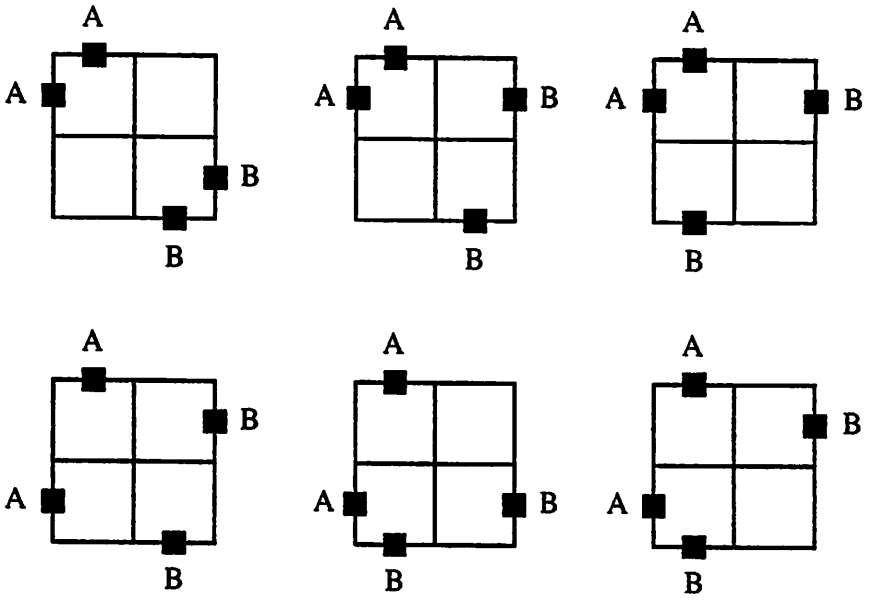


Figure 5.6: *2L-net* pin-position configurations without internal pins

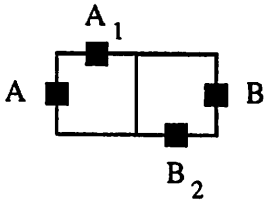
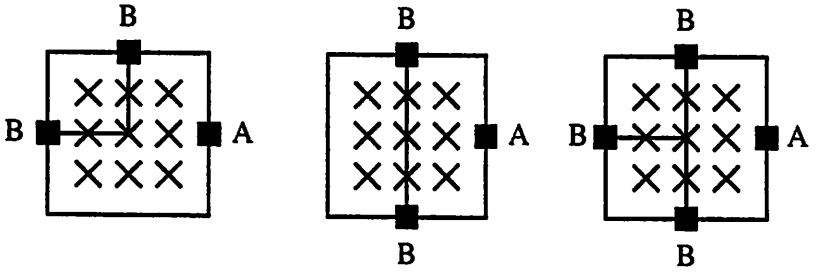


Figure 5.7: *2L-net* case on a 1 × 2 grid

If the *2L-net* problem is too small to be abstracted as a 2 × 2 grid problem, then under the following conditions, the problem still may be routed. If all sides of the switchbox are less than two routing pitches, i.e., the problem is an abstract 1 × 1 grid, then the problem may be routed only if the legal pin layers allow partitioning of the two nets onto separate layers. As in the *2I-net* case, each net is then routed as a single-net problem confined to a particular routing layer. If any one set of parallel switchbox edges is less than two routing pitches in length, i.e, the problem is an abstract 1 × 2 grid, then there are two possible situations where the problem can be routed. First, if the pins on the long sides of the switchbox do not impose any vertical constraints on each other, then the problem can be handled as disjoint single-net problems. This situation corresponds to having a channel routing problem of unity density. An example of this topology is shown in Figure 5.7. Here, net A corresponds to pins A and A_1 and net B corresponds to pins B and B_2. Note that the

Net A has a single edge pin and the internal pin.



Net A has two edge pins and the internal pin.

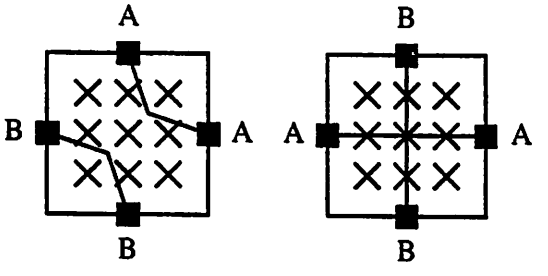


Figure 5.8: Double-net configurations with internal pins

pins A_1 and B_2 may not have been originally specified at the diagramed grid positions, but that their range of legal positions may have allowed enough movement to avoid a vertical constraint. Second, if the first condition does not hold, then again, an attempt can be made to partition the two nets onto separate layers and then route each of them as a single-net problem on a specific layer.

Now consider the effect of including an internal pin in the two-net subproblem. This dramatically increases its complexity. Consider a problem with two nets, A and B. If one of the nets contains an internal pin, the the problem can be classified as shown in Figure 5.8. As in the single-net figures, the X's in Figure 5.8 represent possible internal pin positions relative to the other edge pins of the problem. However, in this figure, all the possible internal pin positions have been shown, and the possible reductions will be discussed later. If net A contains the internal pin, then the following classifications result.

1. Net A has a single edge pin and the internal pin.
   Net B can be either an *I-, L-, or T-net.*

2. Net A has two edge pins and the internal pin.

The two edge pins of A considered separately can be either an *I-net or an L-net.* Thus, net B can be either an *I-net or an L-net*, respectively.

Note that if net A contains three edge pins then net B must contain the internal pin.

Consider the first classification where net A only contains a single edge pin and a single internal pin. If net B is an *L-net*, then the number of pin topologies can be calculated as follows. When abstracting to a 2×2 grid problem, each of the pins of net B will be assigned to 1 of 2 grid rows or columns. Similarly, the edge pin of net A will be assigned to 1 of 2 grid rows, and the internal pin of net A will be in 1 of 4 grid cells. Thus, the number of topologies is $2 \times 2 \times 2 \times 4 = 32$. If net B is an *I-net* then a similar analysis applies, except that problem symmetry allows us to ignore the choice of grid row for the edge pin of net A. Therefore, the number of topologies is $2 \times 2 \times 4 = 16$. If net B is an *T-net*, then all three pins of net B may be assigned to 1 of 2 grid rows or columns, the edge pin of net A may be assigned to 1 of 2 grid rows, and the internal pin will be in 1 of 4 grid cells. However, two of the four permutations of the edge pin of net A and the edge pin of net B on the opposite side of the switchbox are redundant under symmetry. Thus, the number of topologies for this case is $2 \times 2 \times 2 \times 2 \times 4 \div 2 = 32$.

If the problem is too small to form a 2×2 grid abstraction, then it may still be routed under the following special cases. If the problem is an abstract 1×1 grid then the pin layer-specifications must allow partitioning of the nets onto different layers. If the problem is an abstract 1×2 grid then the problem may be routed if it can be separated into independent problems or if the the nets can be partitioned onto separate layers. These conditions are similar to the special cases for *2L-nets* without internal pins. Also, even if the problem is large enough to form a 2×2 grid abstraction, a feasible abstraction may not exist. In this case, there still may be situations that can be routed by detouring net B around the internal pin of net A. This is another special case and an example situation is shown in Figure 5.9. Here, the legal pin layers force net A and net B to be routed on the same layer. The pins of net B cannot be moved, and thus a straight path segment would short net B with the internal pin of net A. Fortunately, the subproblem is large enough to allow net B to detour around the internal pin. In this case, the subproblem could actually be abstracted to a 3×3 grid.

Now, consider the classification where net A contains two edge pins and the internal pin. If net A is an *2L-net*, then the six possible edge pin topologies are the same as shown

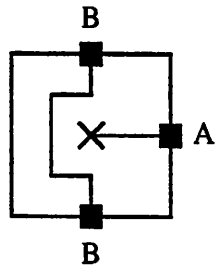


Figure 5.9: Detouring around an internal pin

Class	Count
Single net	
without internal pin	4
with internal pin	17
Two nets	
without internal pin	10
with internal pin	120
Total	151

Table 5.1: Count of basic routing patterns by classification

in Figure 5.6. Since, the internal pin will be in 1 of 4 grid cells, the number of cases is $6 \times 4 = 24$. If net A is an *2I-net*, then the four possible edge pin topologies are the same as shown in Figure 5.5. Again, since the internal pin will be in 1 of 4 grid cells, the number of cases is $4 \times 4 = 16$. The special cases for when the problem is too small to form a 2×2 grid abstraction or for when a feasible abstraction does not exist are similar to the cases described earlier. A summary of the total number of basic cases is given in Table 5.1.

The idea of using a pattern router to perform the final detail routing is attractive because it meshes well with the divide-and-conquer philosophy. An ideal decomposition method should be able to generate subproblems that can be trivially routed. Unfortunately, in practice, the current method produces large numbers of subproblems that can not be routed by the pattern router. To provide more insight on how to improve this process, some data is presented on the characteristics of these subproblems. Table 5.2 shows statistics on these subproblems based on examples run using the best *minimum density* cost function combination for irregular cut paths. The table shows statistics based on number of layers, size, number of nets, and number of internal pins. There are two major trends in the data. The first is the frequency of overflowed subproblems as a function of the number of

Characteristic	Subproblem Type		
	Only Metal 1	Only Metal 2	Two Layer
Frequency (%)	0.0	30.1	69.9
Area (λ^2			
Maximum	0.0	759.0	4745.0
Minimum	0.0	56.0	56.0
Mean	0.0	98.2	160.7
Net Count			
Maximum	0.0	4.0	4.0
Minimum	0.0	2.0	2.0
Mean	0.0	2.1	2.1
Internal Pin Count			
Maximum	0.0	1.0	1.0
Minimum	0.0	0.0	0.0
Mean	0.0	0.0	0.4

Table 5.2: Characteristics of subproblems with overflows

available routing layers and the second is certain characteristics common to all the failed subproblems.

The table shows statistics for subproblems with overflows where the subproblems have been classified into different types depending on the number of available routing layers. For the current examples, there are three possibilities. Either both layers of metal are available for routing or only *metal 2* is available or only *metal 1* is available. In these experiments, it has been found that the majority of subproblems with overflows are two-layer problems. The next largest group are subproblems confined to *metal 2* and the smallest group are subproblems confined to *metal 1*. This trend is evident in Table 5.2. Note that though there were no single-layer, *metal-1* subproblems with overflows for these data, in general, there may be a few. At first, it may seem counter-intuitive that more two-layer problems have overflows than single-layer problems, but there are at least two explanations for this result. First, distribution of subproblems with overflows across layer combinations mirrors the distribution of subproblems in general across layer combinations. Second, the statistics in the table show that more two-layer problems have internal pins than do single-layer problems. Since all the subproblems either have one or zero internal pins, 40% of the two-layer problems have an internal pin, while hardly any of the single-layer *metal-2* problems have an internal pin. As seen in the discussion of pattern enumerations, subproblems with internal pins are more complex and have tighter restrictions on feasible routing

patterns. Thus, it is not surprising that the pattern router fails to route subproblems with internal pins more frequently. Also, for this set of examples, virtually all the blockages are on *metal 1*. Thus, there are very few subproblems confined only to *metal 1*, while there are many subproblems confined to *metal 2*. These single-layer *metal-2* subproblems correspond to over-the-cell routing problems.

The number of nets and size in area of subproblems are perhaps the most telling statistics in the table. The final, target subproblem is defined as having at most two nets, yet all of the overflowed subproblems have at least two nets. Thus, the subproblems are either beyond the scope of the pattern router or impossible to route. The small size of these failed subproblems compounds the problem. For reference, in the technology of the current examples, the *metal 1* pitch is 7λ and the *metal 2* pitch is 8λ. Thus, the smallest 1×1 grid abstraction has an area of $56\lambda^2$ and the smallest 2×2 grid abstraction has an area of $224\lambda^2$. The data in the table shows that the average size of a subproblem with overflows is smaller than a 2×2 grid. Except for a few special cases, subproblems of this size are impossible to route.

The data indicate that the hierarchical decomposition process is not doing enough to solve the routing problem at higher levels in the hierarchy and/or reduce the complexity of subsequent levels. Instead, the process appears to postpone dealing with the complexities of the problem by continually pushing the real problem down into the subproblems. This ultimately results in subproblems that can not be routed and is shown in the data by minimum size subproblems with large numbers of nets. While one might expect this behavior from a minimum density cut path cost function heuristic, there must be a more basic explanation since this particular cost function combination produced the fewest number of total overflows among all irregular cut path heuristics. One interpretation is that the two-partition model and the pin assignment model do not capture enough of the essence of the actual routing problem. For example, using these models it is difficult to construct cost functions that will handle properly cyclic constraints in a simple channel. The following improvements are suggested for the hierarchical decomposition process described in this report. First, use a four-partition model rather than a two-partition model. Net abstractions must deal with at least two dimensions and the four-partition model provides a more natural structure for analyzing the effects of bends in path segments and the interaction of nets competing for space across an area. Second, use a more complex pin assignment model or at least more complex pin assignment cost functions. The current cost functions

are quite simplistic and optimize basically for net length without considering subsequent congestion and blocking effects. Also, one could consider stopping the decomposition at larger subproblems and then using a more sophisticated algorithm than pattern routing to perform the final detail routing. Unfortunately, this runs counter to the goal of a consistent hierarchical decomposition method that spans the spectrum of global and detail routing. However, this may be the best solution in terms of the engineering trade-offs in the implementation of the method. In the current implementation, run times are dominated by the partitioning process.

Another improvement to the decomposition model deals with its coupling to the pattern router. The analysis of a final subproblem is not really a simple pattern match and allows for many special cases. The dominant factor in this complexity is the part of the partitioning model that treats pseudo-pins and internal pins differently. As seen in Table 5.1, internal pins contribute overwhelmingly to the number of basic patterns. Since internal pins can not be avoided in the final subproblems, the treatment of pseudo-pins must be changed and made consistent with the treatment of internal pins. That is, each current pseudo-pin should represent two pseudo-pins, one for each side of the partition. In this model, pseudo-pins lie on the same abstract grid as the internal pins and thus, all the pins in the problem are treated consistently. With out the distinction between pseudo-pins and internal pins, the patterns in the enumeration reduce to a set similar to the set of cases without internal pins. The layer specification of the path segment implicitly connecting the two pseudo-pins is restricted by the layer constraints of what used to be the single pseudo-pin and independent layer changes may occur at each of the two pseudo-pins if necessary. Also, note that the specification of the target final subproblem was derived as a matter of convenience for the subsequent case enumeration. The relationship between the characteristics of subproblems and the partitioning process was not fully considered and the target final subproblem is not often a the natural result of the partitioning process. This is an aberration to the goal of a consistent decomposition process. Again, since 2×2 abstractions seem most usable, a switch to a four-partition model should provide more natural and consistent subproblem formulations.

5.2 Maze Routing

In early experiments, a maze router was used to perform the final subproblem routing. For these experiments the partitioning was stopped when a subproblem contained only two nets and the size of the routing region decreased below some maximum upper bound. The size constraint was introduced to prevent the program from trying to maze route subproblems that could be routed faster if they were partitioned further. In retrospect, the maze router probably performed better than the pattern router. The switch to the pattern router occurred on the premise that it would provide a speed advantage over the maze router. Unfortunately, the extra partitioning required by the pattern router is past the break-even point for the current implementation of the method and the run time of the program using the pattern router is dominated by the partitioning time. More work needs to be undertaken on when to stop the partitioning process. For the current implementation, our results indicate that it should be stopped at a higher level than the simple switchboxes required by the pattern router. The actual stopping criteria will depend on the partitioning model and its implementation as well as the subproblem routing method.

5.3 Subproblem Routing Order

The subproblems in the method described in this report are not independent of each other in two ways. First, the pseudo-pins are allowed to float at a fixed position along their respective switchbox edges until constrained by detail routes. Second, within the restrictions of the containing partition segment, the pseudo-pin layer assignments also remain unconstrained until specified during the final subproblem routing. The effect of these to characteristics is to make adjacent subregions mutually dependent and thus the order of routing the subproblems affects the final solution.

Some work was done on this subject by Nishizaki [Nis] using an early version of the program written for this work. Based on related experiments and his results, the following priority function has been chosen to order the subproblems.

$$priority = 30(L - l) + 10n + p$$

where

Percent (%) Reduction	Irregular Cut Path Cost Functions	
	Minimum Capacity	Minimum Density
Maximum	15.9	13.7
Median	11.9	12.6
Mean	11.9	11.3
Minimum	8.0	6.5

Table 5.3: Percent reduction in overflows due to ordering heuristic

L = the maximum number of routing layers in the design
l = the number of routing layers in the subproblem
n = the number of nets in the subproblem
p = the number of pins in the subproblem

Table 5.3 shows statistics on the percentage reduction in the number of total overflows that results from using this priority function instead of a depth-first search of the hierarchy. Values were obtained for both the best *minimum capacity* and the best *minimum density* cost function combinations. Based on the table, using the priority function heuristic always performs better than using the depth-first search order.

Nishizaki's results are based on a version of the program that used a maze router to perform the final subproblem routing. In addition to static ordering, he also tried ordering methods based on dynamic calculation of priorities, adjacency graphs, and bottom-up clustering. However, the simple static ordering method performed best. Some other possible variations include combining a dynamic ordering scheme with rip-up and re-route phases and applying an ordering method to nets instead of or in addition to subproblems. For example, groups of nets could be routed in order according to their priorities. The priority values could be either predefined by the user or calculated dynamically by the program. For each group of nets, all the subproblems containing a net or nets from the group would be routed using the chosen subproblem ordering technique. These more detailed experiments in subproblem routing order are left for future work.

Chapter 6

Rip-up and Re-route

The routing problem has been shown to be NP-complete [Joh83] and as such, any heuristic constructive approach can not guarantee 100% success on all problems. The concept of rip-up and re-route has been used extensively in the past (particularly in gate-array routing) to resolve overflows by providing a method of backing out of the bad decisions made by the heuristics. A few recent examples of the use of rip-up and re-route techniques are [DK82, MS84, SSV87, TS88]. Here, the goal is to implement a rip-up and re-route algorithm suitable for the hierarchical structure of the decomposition described in this report.

The idea behind rip-up and re-route techniques is to provide an iterative back-tracking process that will allow the routing program to escape dead ends in the routing solution space. Typically, there are two kinds of modification performed by rip-up and re-route techniques. The first kind of modification moves or relocates previously routed nets or net segments to make room for a blocked net or to allow a shorter path for a net. This has been called *shove-aside* [DK82] or *weak modification* [SSV87]. The second kind of modification removes previously routed nets or net segments to allow the connection of a blocked net. Nets that have been ripped-up are put in a list for later re-routing. This has been called *rip-up* [DK82] or *strong modification* [SSV87].

The following describes how rip-up and re-route techniques may be applied to a tree of hierarchical subproblems.

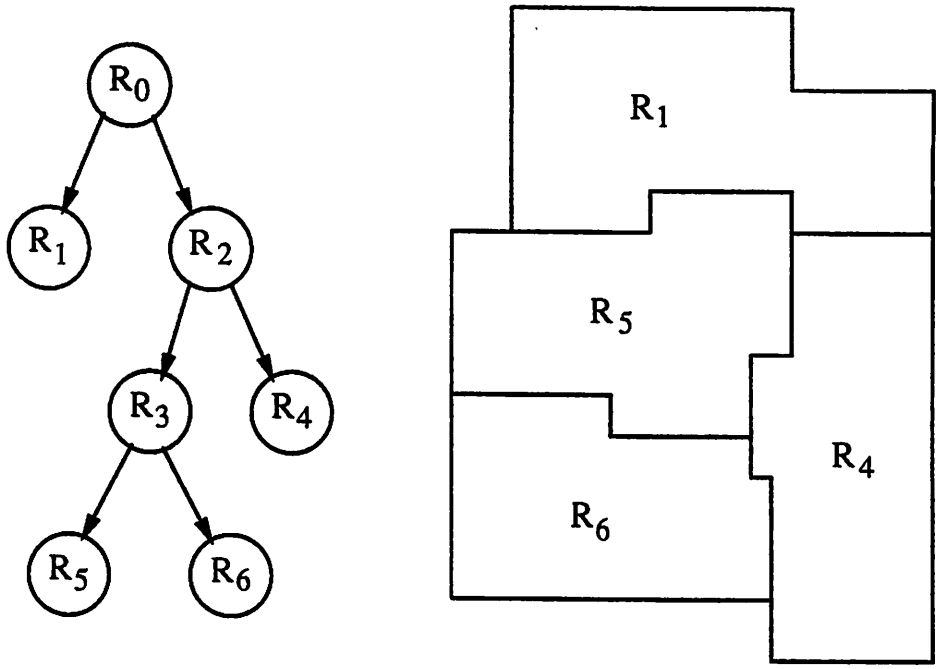


Figure 6.1: Relationship between subproblem tree and partitioned region

6.1 Hierarchical Rip-up and Re-route

The application of rip-up and re-route techniques to a hierarchy of subproblems requires considering two situations. These cases are ripping-up and re-routing nets to relieve congestion within a subproblem and ripping-up and re-routing nets to relieve congestion on the boundary between sibling subproblems. To provide background, the relationship between the tree of hierarchical subproblems and the partitioned routing region will be reviewed, some terms will be defined, and some general observations about changing routes in the hierarchy will be discussed.

The hierarchical two-partitioning performed by the partitioning process in this report produces a binary tree of subproblems. The relationship between this tree and the partitioning of the routing region is shown in Figure 6.1. The root of the tree corresponds to the original routing problem. The children nodes of the root node correspond to the two subproblems (and thus the two subregions) created by the partitioning. The partitioning process is recursive and thus this relation holds for all nodes and their children. A leaf node corresponds to a final subproblem. In Figure 6.1, R_0 is the root of the routing hierarchy and corresponds to the original routing problem. This problem was partitioned into R_1 and

R_2. The subproblem R_2 was partitioned into R_3 and R_4 and finally the subproblem R_3 was partitioned into R_5 and R_6. The leaf nodes of the subproblem tree are R_1, R_4, R_5, and R_6 and are the final subproblems of the hierarchical decomposition that must be routed.

Now, consider how overflows may occur with respect to the problem hierarchy. The term *partition overflows* is used to refer to overflows caused by insufficient capacity along a partition and the term *subproblem overflows* is used to refer to overflows that occur within a subproblem. Note that a partition overflow between two siblings can be considered a subproblem overflow of the siblings' parent.

The actual path of a given net is gradually refined throughout the hierarchical process. At each step more constraints are placed on its route by using pseudo-pins to mark where it may cross subregion boundaries. A *feed-through net* is defined as a net of a subproblem that does not contain any pins of the original routing problem, i.e., it only contains pseudo-pins.

At each stage in the decomposition, subproblems are partitioned into their constituent subproblems. The cut path that physically delineates this partition forms the common boundary of the constituent subproblems and is referred to as the partition of the subproblem it partitions.

To motivate the following discussion, consider how routes may be changed in the hierarchy. Figure 6.2 shows a routing region, the corresponding routing hierarchy, a global route of a particular net in that hierarchy, and a modified route of the same net. In the routing region, the solid line shows the original path of the net and the broken line shows the modified path. Nodes in the routing hierarchy are shaded if the corresponding subregion contains the net. Two versions of the routing hierarchy are exhibited, corresponding to the original and modified versions of the net's path. The path of the net may be changed from passing through both of a pair of sibling subregions to passing through only one of the subregions or vice versa by changing the pin position assignment on a partition of one of the siblings' ancestors. In this particular case, the net is changed from traversing both of the sibling regions R_{13} and R_{14} to traversing only R_{13} by changing the pseudo-pin position on the partition of R_2. Thus, pin reassignment in the hierarchy is an example of the rip-up and re-route technique known as weak modification. Later, another operation will be discussed that also performs weak modification.

Notice that the pin reassignment on R_2 changes the subproblems corresponding to R_5 and R_6. These changes must be propagated throughout the trees rooted at R_5 and

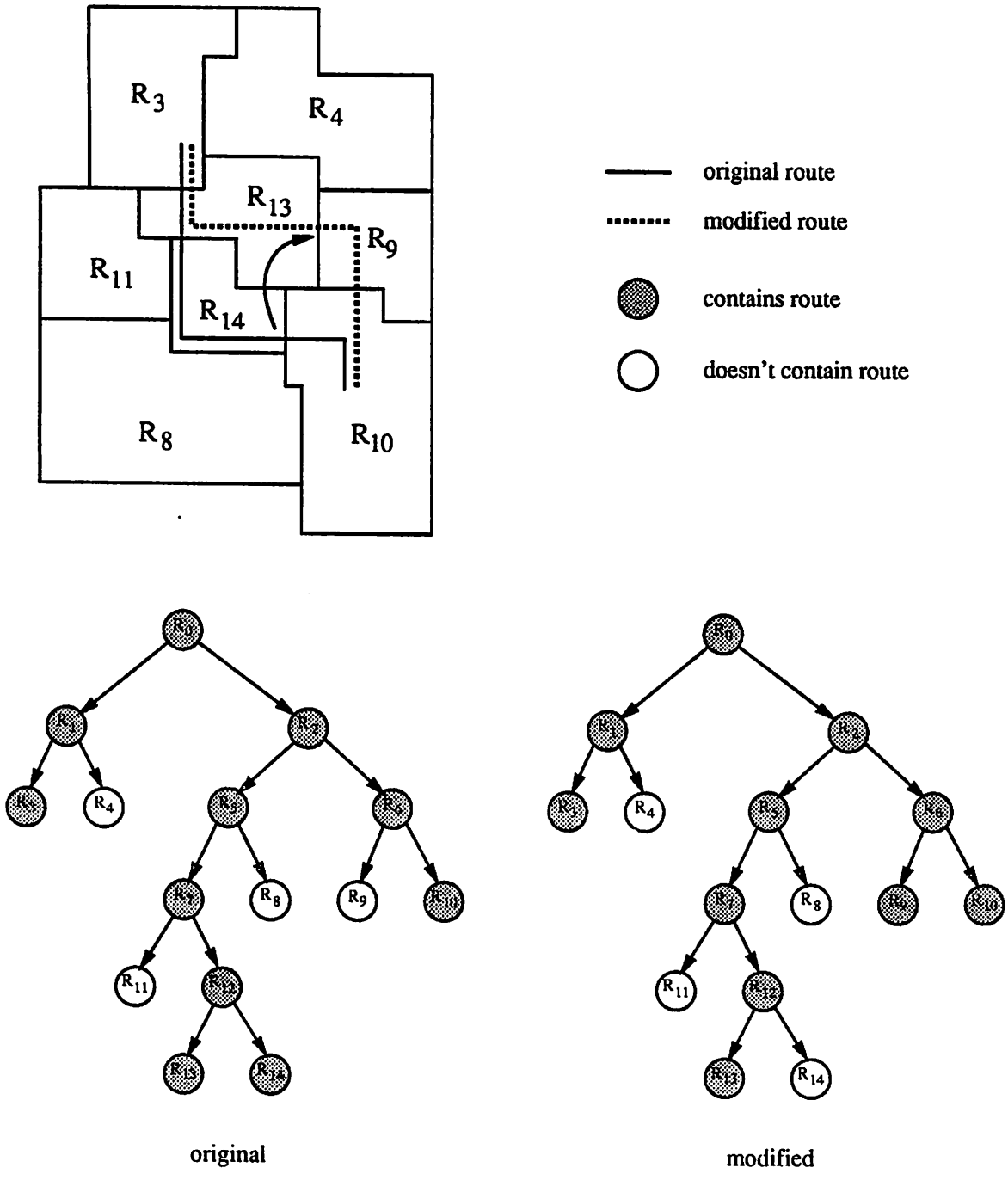


Figure 6.2: Route modification in the region hierarchy

R_6. These subtrees must be traversed to determine whether the changes create any other overflows. I.e., the modification is legal only if the changes don't cause one of the moved nets to overflow in some other region. Thus, it is desirable to perform the pin reassignment at the root node of the smallest possible subtree in order to minimize the amount of required change propagation checking. This criteria is also optimal in the sense that a minimal-size net segment will be moved to effect the re-route.

6.1.1 Subproblem overflows

Based on the previous discussion about moving nets between sibling subregions, the problem of rip-up and re-route in a subproblem might be defined as follows. Given a subproblem that can not be routed because of some particular congested area and given the subset of nets of the subproblem that cross this congested area, change the routes of enough of these nets to make routing across the congested area feasible.

From previous observations, a simple way to handle this situation is to completely remove any feed-throughs that pass through the congested area from the subproblem. In the following, different feed-through topologies are examined and the effect on the rest of the hierarchy of re-routing the feed-through nets is discussed.

The simplest situation occurs when one pseudo-pin of the feed-through net is on the partition of the subproblem's parent and the other pseudo-pin is on the partition of the subproblem's grandparent. This case is illustrated in Figure 6.3. Here, R_{13} is congested and this congestion can be relieved by moving a portion of the net shown. The net segment that will move appears as a feed-through net in R_{13}. The feed-through net has one pseudo-pin on the partition of R_{12}, R_{13}'s parent region, and one pseudo-pin on the partition of R_7, R_{13}'s grandparent region. By performing a pin reassignment on the partition of R_7, the route of the net can be changed to traverse only R_{14} instead of traversing both R_{13} and R_{14}. This relieves the congestion in subproblem R_{13} by completing removing one of its nets. In general, the partition of a subproblem's grandparent does not necessarily form a boundary of the subproblem, but this is a common case. Performing a pin reassignment on the grandparent's partition allows us to remove the net from the subproblem and isolate it in the subproblem's sibling. Of course, this is dependent on there being sufficient excess capacity along the grandparent's partition to allow the reassignment. Note that this case corresponds to the smallest amount of required propagation in the hierarchy since the grandparent's partition

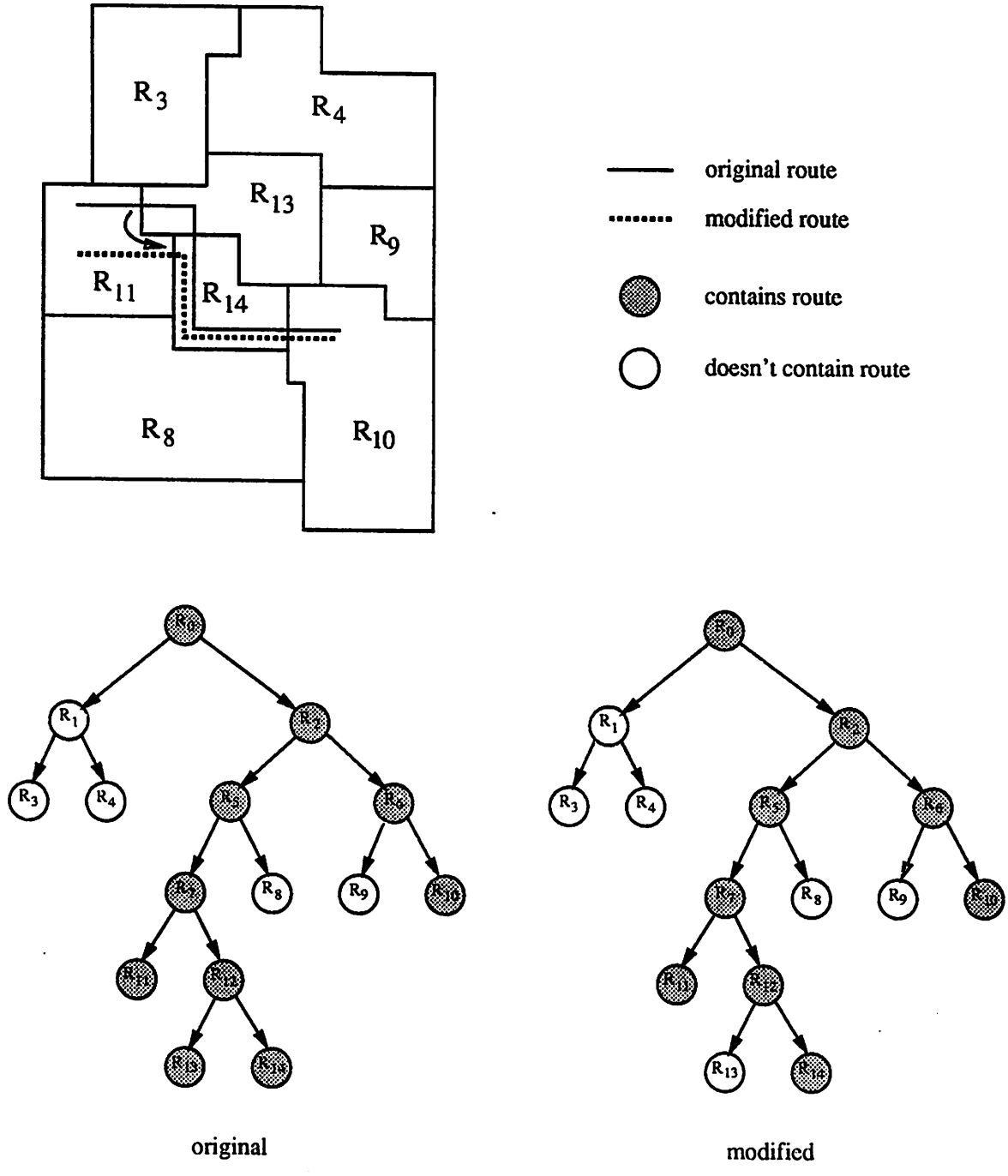


Figure 6.3: Parent-grandparent re-route topology

is the closest partition in the subproblem tree at which a reassignment can occur. In general, the feed-through net may have any number of pseudo-pins as long as they all fall on either of the two desired partitions. A feed-through net with only two pseudo-pins is shown for simplicity. In practice, the number of pseudo-pins will probably have been reduced to two by previous efforts to relieve routing congestion.

The next topology allows the pseudo-pins of a feed-through net to be on the subproblem's parent partition and on a partition of any ancestor of the subproblem that intersects both the subproblem's boundary and the subproblem's sibling's boundary. An example of this situation is shown in Figure 6.4. The sibling regions are R_{13} and R_{14}. The parent region is R_{12} and the ancestor region is R_2. As in the simple case, the idea here is to route the feed-through net completely in the subproblem's sibling by doing a pin reassignment on an ancestor's partition. Unfortunately, this may not be a very local modification. If the chosen partition is the partition of the original routing problem, then the pin position changes potentially must be propagated through the entire design. In this example, the entire subtree rooted at R_6 has been modified by re-route. Luckily, if the modifications really are local, the propagation of the change in the hierarchy can be pruned easily. For example, suppose that R_8 has been decomposed further so that it is the root of a subtree of the routing hierarchy. Now, apply the example re-route. When the propagation reaches R_8, it can be determined that the region originally did not contain the net and that it still does not contain the net. Thus, the propagation does not need to traverse the subtree rooted at R_8.

The feed-through re-route model can be generalized even more by allowing all the pseudo-pins of the feed-through to be on any two ancestor partitions. As before, each partition must must intersect the boundaries of both the current subproblem and its sibling. An example of this case is shown in Figure 6.5. The sibling subregions are R_{13} and R_{14} and the ancestor regions are R_2 and R_7. In this situation, the objective is still to isolate the route of a net in a particular sibling subproblem. The difference is that the feed-through could have started being completely routed in the other sibling. This is the case shown in the example. Originally, the net segment is routed only in R_{13}, but after the re-route, it is routed only in R_{14}. Like the parent-ancestor topology, implementing this case may require a lot of change propagation. Unlike the previous examples, this topology may require pin reassignments on two partitions rather than on just one. In the example, pin reassignments are required the partitions of both R_2 and R_7.

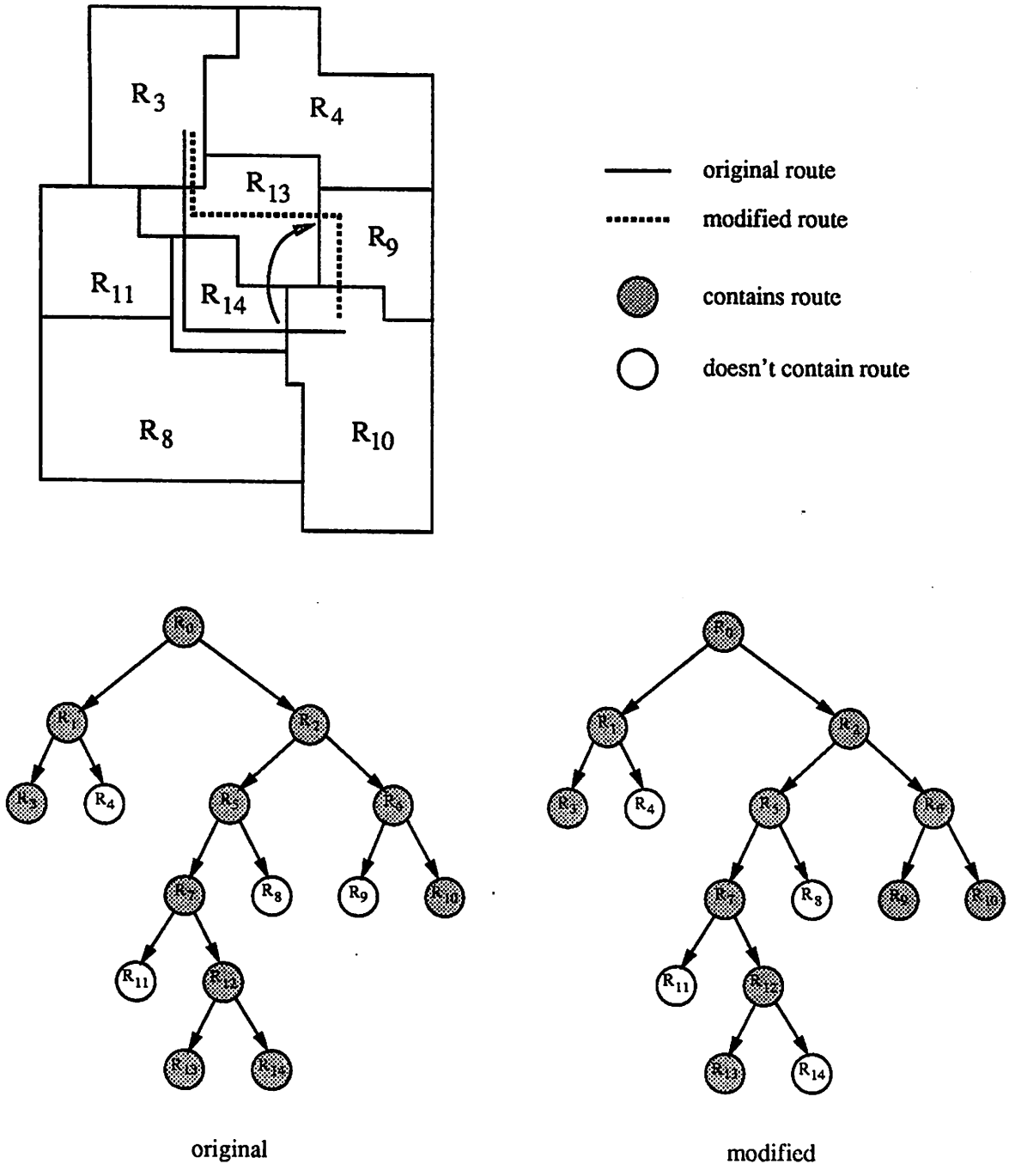


Figure 6.4: Parent-ancestor re-route topology

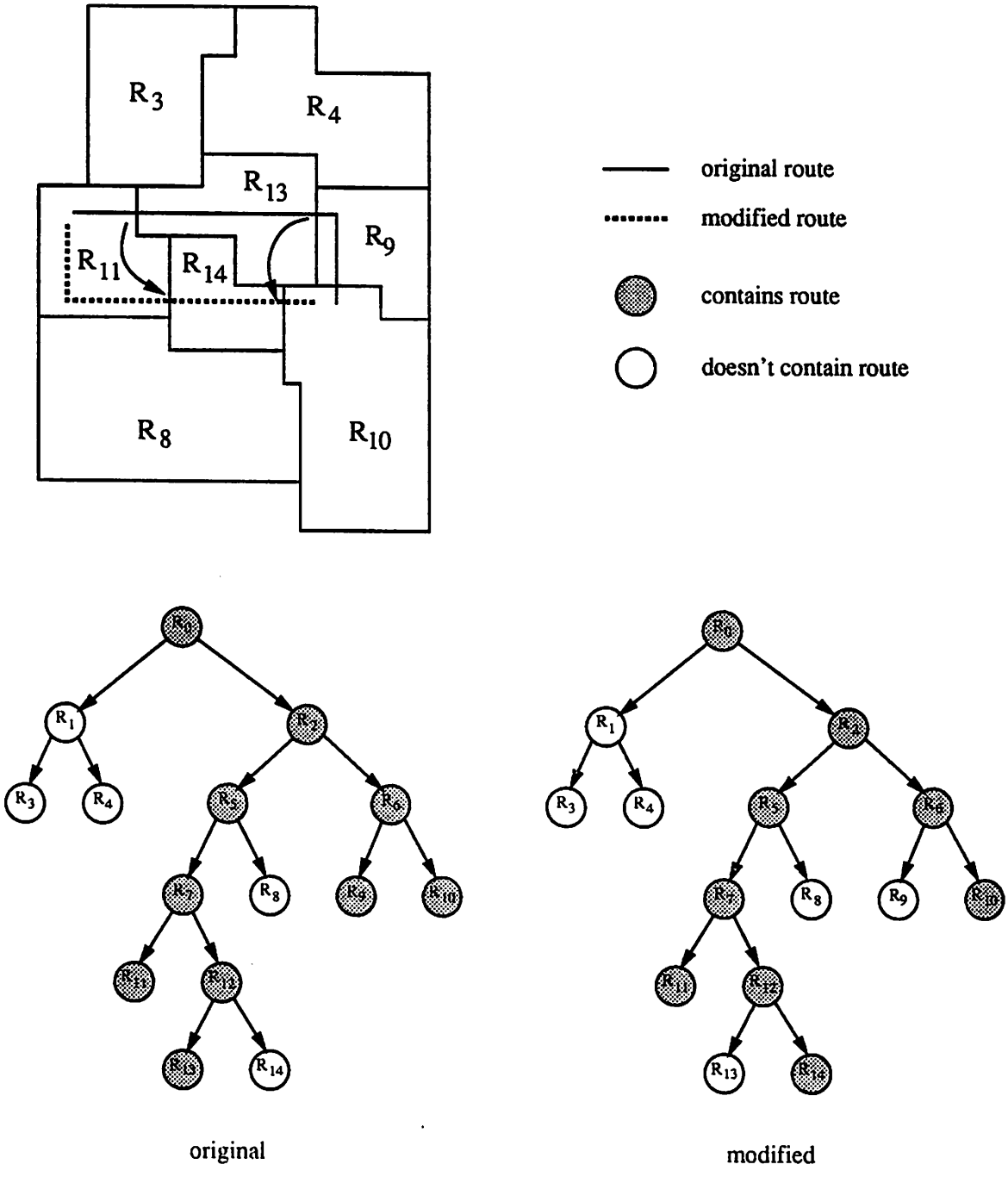


Figure 6.5: Ancestor-ancestor re-route topology

As mentioned earlier, it is possible to perform weak modification without restricting the re-route to feed-through nets. Consider the situation in Figure 6.6. The subproblem R_{14} is congested and the nets that want to cross the congested area are not feed-throughs and can not be moved by any of the above pin reassignments. However, if pseudo-pins are added to a boundary of the subproblem, the net can be made to detour outside of the subproblem. In the example, the common boundary with regions R_{11} and R_8 is chosen. This requires having a boundary with excess capacity, a priori knowing where the congestion is so as to pick the proper boundary and pseudo-pin locations to avoid it, and enough capacity in the adjacent subproblem(s) to route the detour. Except for the special case where the congestion is due to partition overflows between descendants, there is no easy means of determining the nature of the local congestion and how to pick the appropriate boundary and pseudo-pins to avoid it. This partition overflow case will be discussed in Section 6.1.2.

The feed-through re-route scheme can be applied hierarchically. If a net can not be moved from a subproblem to its sibling, then the process can be applied at the next higher level to try to move the net from the parent of the subproblem to the parent's sibling. An example of this is shown in Figure 6.7. The original subregion siblings are R_{13} and R_{14}. Subproblem R_{13} is congested and the first re-route attempt fails because the partition of R_7 has insufficient capacity along the border of R_{14} to allow a pin reassignment that would remove the net segment from R_{13}. However, at the next higher level in the hierarchy, the parent-grandparent topological model is re-applied and a net segment is moved from R_{12}, R_{13}'s parent, to the sibling region, R_{11}, by a pin reassignment on the partition of R_5. This ultimately removes a net segment from R_{13} and relieves the congestion. Note that the hierarchy provides a mechanism for minimizing the impact of the re-route process. The process starts by trying to change a minimal-size net segment to perform the modification and then gradually uses larger and larger net segments as it backtracks up the hierarchy. Thus, with respect to the granularity of the hierarchy, the minimal amount of required re-route is used to relieve congested areas.

6.1.2 Partition overflows

Now, consider the case of applying rip-up and re-route to resolve partition overflows. The problem is defined as follows. Given a partition that has insufficient capacity to accommodate all the nets that must cross it, modify the routes of enough of the crossing

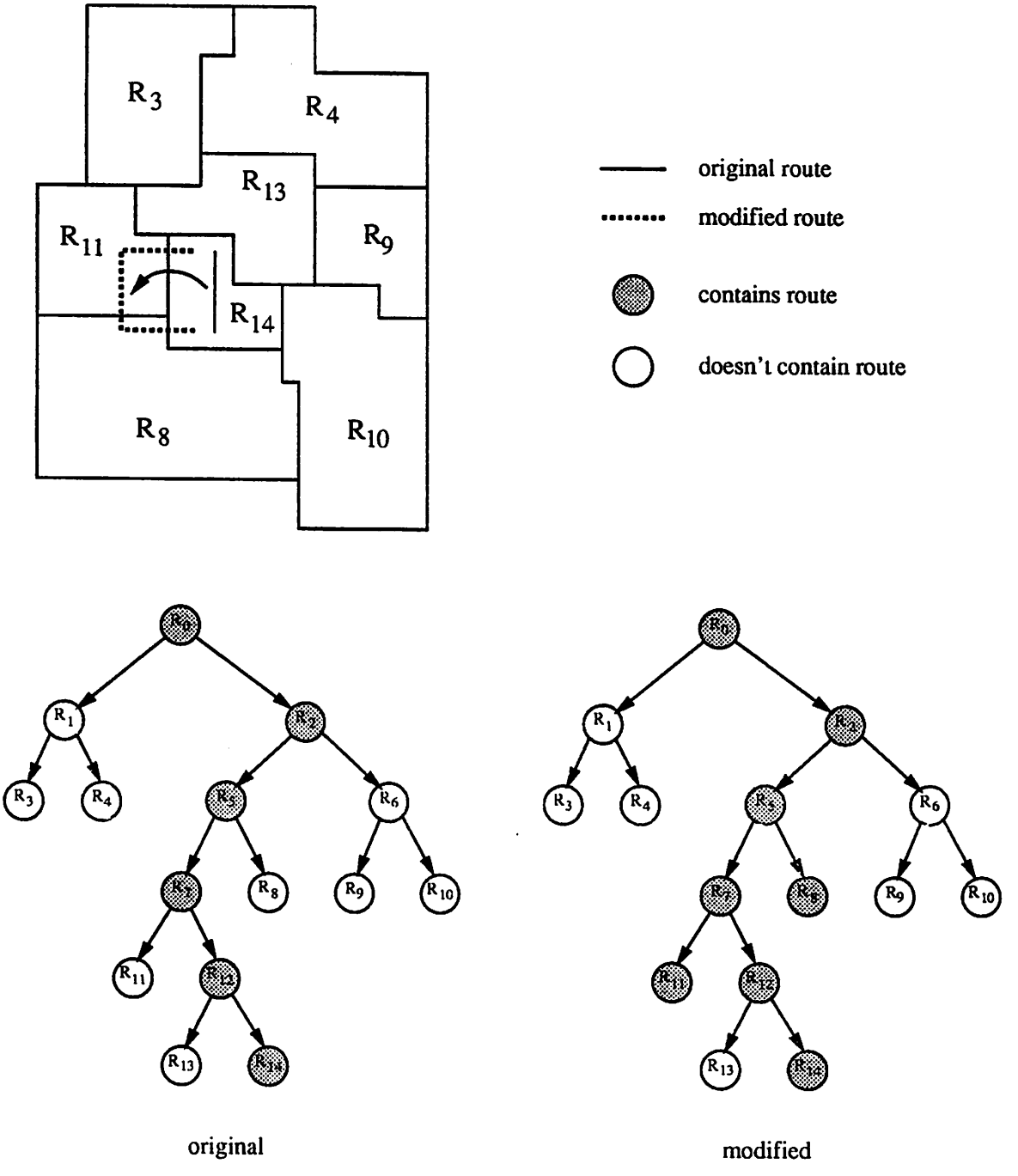


Figure 6.6: A complex detour modification

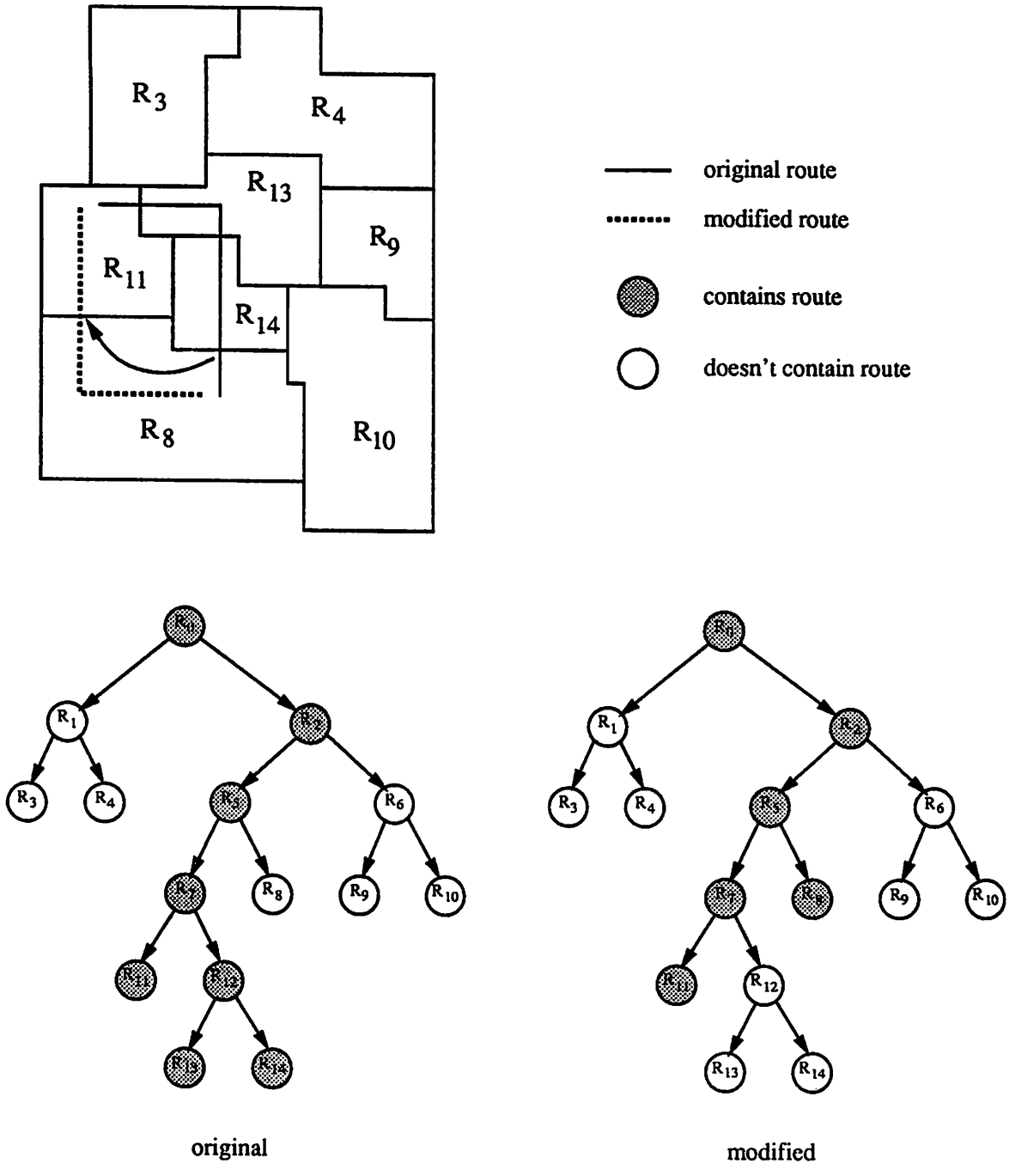


Figure 6.7: Hierarchical application of the feed-through re-route modification

nets so that they no longer cross the partition and so that the remaining crossing nets can be routed across the partition. Basically, the same conditions hold as in the subproblem re-route case and in fact, after the first step, the sub-problem re-route technique may be used exactly.

For the subproblem re-route case, one possibility is to move a parent-ancestor feed-through net so that a net that originally traversed two sibling regions only traverses one of the siblings. In effect, this changes the route of the net so that it no longer crosses the parent's partition. Changing the route of a net so that it no longer crosses a particular partition is the goal of resolving partition overflows and thus partition overflow re-route techniques will utilize the same topologies as discussed in the last section.

The difference between partition overflow re-route and subproblem overflow re-route is that both siblings must be considered at once and thus, the subproblem re-route can not just be applied sequentially to both sides of the partition. To see this consider the situation in Figure 6.8. Both sibling regions, R_{13} and R_{14}, have feed-through nets which may be re-routed to relieve congestion across the partition of R_{12}. For subproblem R_{13}, this is net A, and for subproblem R_{14}, this is net B. To re-route these feed-through nets requires a pin reassignment on the partition of R_7. Now suppose that the portion of the partition of R_7 that intersects the boundaries of R_{13} and R_{14} has zero excess capacity. Trying to re-route this situation by performing the subproblem technique separately on each sibling fails because the pseudo-pins in each of the subproblems can not move with the other still in place. A successful pin reassignment on the partition of R_7 requires swapping the pseudo-pin positions of nets A and B. Thus, both R_{13} and R_{14} must be considered when performing the pin reassignment on the partition of R_7. In general, the choice of partitions and feed-throughs is the same as for subproblem re-routes, but the pin reassignment must account for a potential swap move instead of just a simple relocation.

As in the subproblem case, if the re-route fails at this level of the hierarchy, it can be re-tried one level higher in the subproblem tree. However, at the level above a partition overflow, the problem transforms into a subproblem overflow problem and the techniques of the previous section may be applied directly for the current level and for all higher levels in the hierarchy. The problem transformation can be seen as follows. The subproblem in question is the subproblem of the partition that overflowed. The subset of nets that are candidates for removal are all the nets that cross the overflowed partition. The number of nets that must be removed is equal to the number of nets that cross the partition in excess

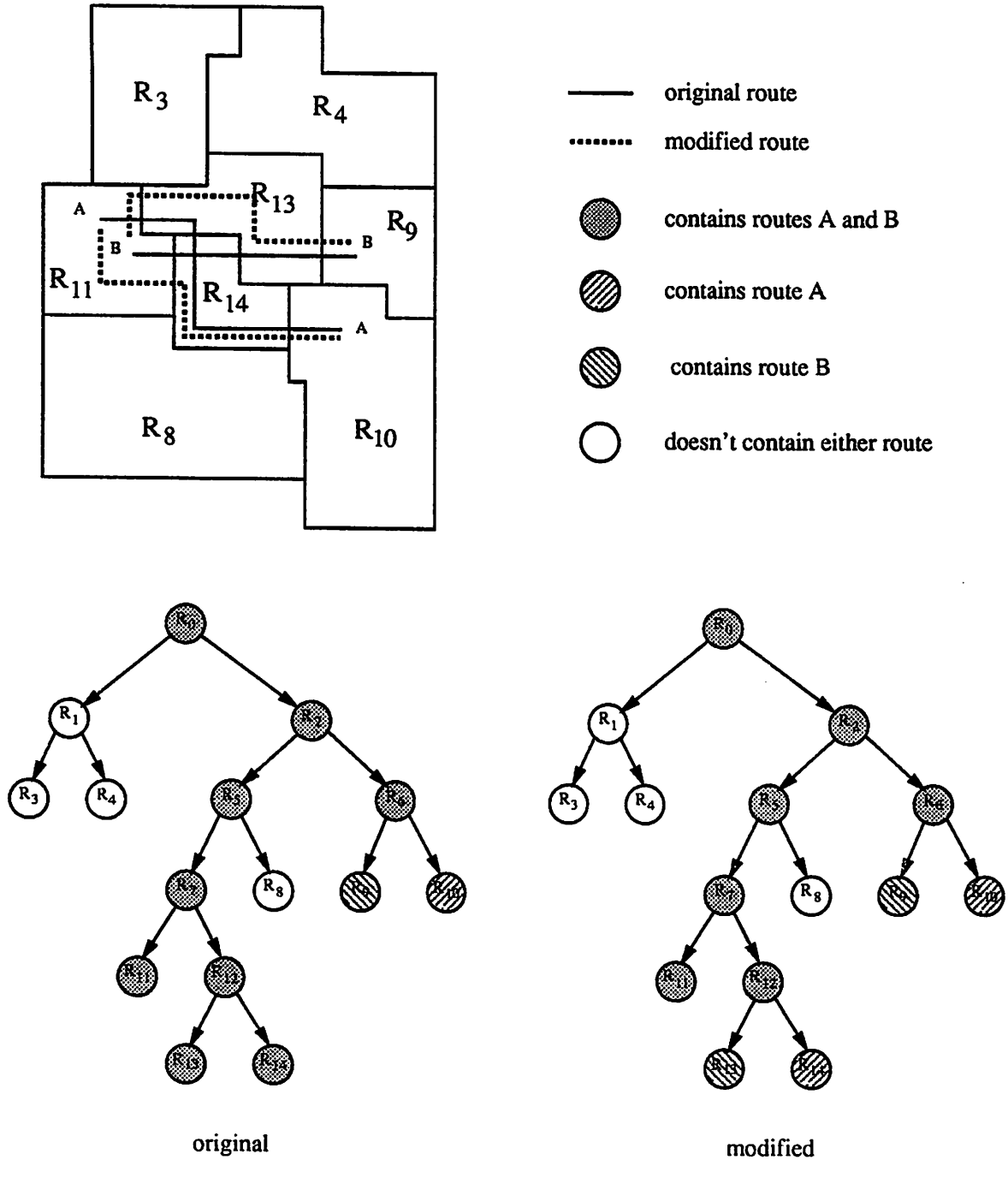


Figure 6.8: Route modification for partition overflows

of the capacity of the partition.

The problem transformation from partition overflow to subproblem overflow reveals the special case where enough information exists to apply the subproblem re-route detour variant described in the previous section. The congestion that must be avoided is represented by the overflowed partition and the possible locations for new pseudo-pins are bounded by the boundaries of the sibling subregions. Figure 6.9 illustrates the two possible choices of detour directions corresponding to detouring around either end of the overflowed partition. In this example, the overflowed partition is the partition of R_{12} and in terms of the description in the previous section, the detour technique is being applied to subproblem R_{12}. As in the feed-through re-route model, the partition that will be modified must coincide with a portion of the boundaries of both children regions, R_{13} and R_{14}. To detour to the left, two pseudo-pins must be added to the partition of R_7. One of these pseudo-pins must be placed on the portion of the partition of R_7 that intersects the boundary of R_{13} and the other pseudo-pin must be placed on the portion that intersects the border of R_{14}. Subsequently, a new net must be added to R_{11} to represent the detour portion of the route. Finally, the changes must be propagated through the subtree rooted at R_7. Similarly, to detour right, two pseudo-pins must be placed on the appropriate sections of the partition of R_2, a new net, representing the detour segment, must be added to R_6, and the effects of the modification must be propagated through the subtree rooted at R_2

6.2 Results

In these experiments, re-routing partition overflows has been investigated using the feed-through re-route model on the parent-grandparent and the parent-ancestor topologies. Subproblem overflows (that do not originate as partition overflows) were not handled because these overflows only occur within final subproblems and in general, can not be detected without actually attempting to route the subproblem. Certainly, specific tests can be applied to check for subproblem overflows. For example, for the pattern router, if a subproblem has more than two nets, then the subproblem will overflow. Also, heuristics maybe used to predict subproblem overflows. For example, certain combinations of pin layer-specifications and subproblem size may produce a high percentage of overflows and thus a heuristic may be formulated to check for these conditions. These experiments have been left for future work. Since final subproblem overflows are not processed, it would

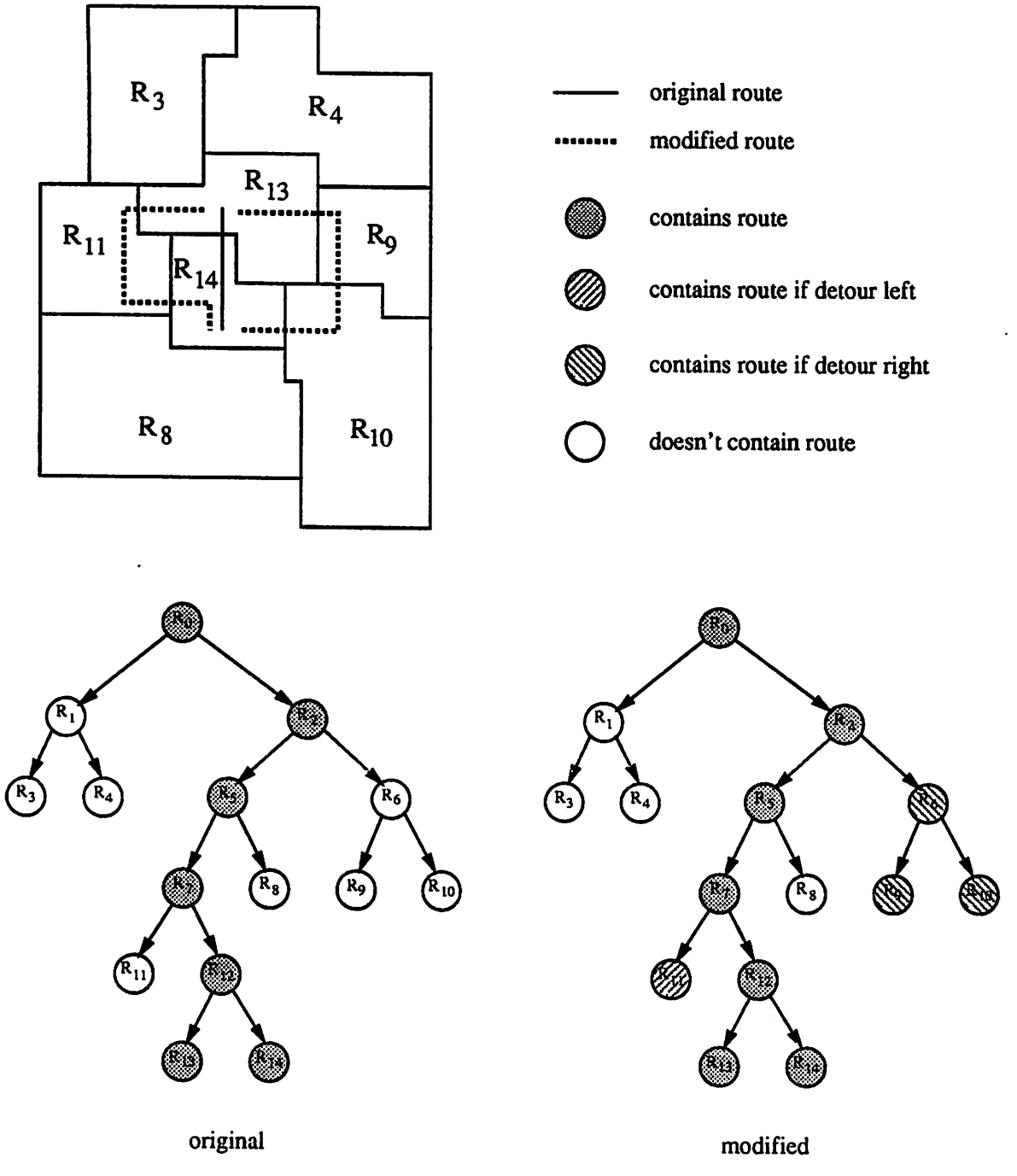


Figure 6.9: Route detour around a partition

not be surprising if the re-routing of partition overflows increases the number of final subproblem overflows, and indeed, this is the case for the current examples. However, since it is possible to implement re-route for subproblem overflows, the increase in subproblem overflows under the current implementation can be overlooked. The point of the current experiments is that rip-up and re-route on the routing hierarchy can resolve a significant number of partition overflows with a reasonable amount of backtracking. Since the re-route of partition overflows and subproblem overflows in the routing hierarchy is consistent, this result should extend to both kinds of overflows in a system that handles final subproblem overflows.

When using rip-up and re-route techniques, some limit must be placed on the amount of backtracking to prevent the method from trying to search all the possible alternatives and consuming too much time. In the current approach, there are several ways to limit the search of alternatives. First, for each re-route attempt, the group of nets to be re-routed are selected only once. That is, a group of candidate nets are selected heuristically and if the re-route fails no other group of nets is tried for the current re-route topology on the current subproblem. The selection heuristic is to prefer feed-through nets that belong to nets that do not have any internal pins in the parent region of the current subproblem. This is based on the belief that it is easier to detour a section of a net that is far away from a pin of the net than it is to detour a section close to a pin. Second, pseudo-pins are reassigned along partitions subject to the constraint that a pseudo-pin may not be moved to a partition segment from which it has been moved previously. Note that as a practical consideration, this restriction also prevents oscillation between different re-route alternatives. Finally, the amount of searching is also limited by the choice of re-route topology models. Initially, only the parent-grandparent feed-through topology model was tried. This is the simplest possible modification and restricts the changes to the smallest possible subtrees. In fact, since a breadth-first decomposition of the routing problem is used, if a re-route attempt using this model works on the first try, then the maximum height of the subtrees to which changes must be propagated is two. However, in general, if the re-route attempt fails, the router tries to apply the topology re-route model to the next higher level in the hierarchy to resolve the overflow.

Table 6.1 shows the percentage of partition overflows that could be resolved by using just the parent-grandparent feed-through model. Results are shown for both *minimum capacity* and *minimum density* cost combinations. The table shows that even this simple

Cost Function	Examples			
	adder2	adder4	adder8	adder2x8
Minimum Capacity	46.7	34.1	31.9	27.9
Minimum Density	37.5	28.6	30.7	15.0

Table 6.1: Percentage of partition overflows resolved using the parent-grandparent feed-through model

Cost Function	Examples			
	adder2	adder4	adder8	adder2x8
Minimum Capacity				
Maximum	8.0	6.0	10.0	13.0
Mean	2.9	2.3	2.2	2.0
Median	2.0	2.0	1.0	1.0
Minimum	1.0	1.0	1.0	1.0
Minimum Density				
Maximum	8.0	21.0	10.0	14.0
Mean	3.2	4.0	2.0	2.7
Median	1.0	3.0	1.0	1.0
Minimum	1.0	1.0	1.0	1.0

Table 6.2: Amount of backtracking using the parent-grandparent feed-through model

case helps significantly — 15% to 46.7% of the partition overflows were resolved using this method.

Table 6.2 shows some statistics on the amount of backtracking in the hierarchy that was required to resolve overflows using the parent-grandparent feed-through model. The values in the table are given as the difference in levels between the region containing the overflowed partition and the region containing the partition on which a pin reassignment fixed the overflow problem. The root region is assigned level 0 and all subsequent regions are assigned consecutive integer numbers according to their topological order. Thus, a value of unity indicates that the re-route worked on the first attempt. Again, both *minimum capacity* and *minimum density* cost combination results are shown. The key result from this table is that of the re-routes that succeeded, most of them succeeded on the first try. Note that both the median and the mean are low compared to the maximum. In practice, a limit can be placed to the maximum number of backtrack levels to prevent the re-route from making too many attempts and to limit the range of the changes that must be propagated.

Since, the amount of backtracking is small for the parent-grandparent feed-through

Cost Function	Examples			
	adder2	adder4	adder8	adder2x8
Minimum Capacity	85.7	72.5	72.4	73.8
Minimum Density	75.0	78.2	72.8	68.9

Table 6.3: Percentage of partition overflows resolved using the parent-ancestor feed-through model

Cost Function	Examples			
	adder2	adder4	adder8	adder2x8
Minimum Capacity				
Maximum	9.0	7.0	25.0	24.0
Mean	3.9	3.1	4.1	4.4
Median	3.0	3.0	3.0	3.0
Minimum	1.0	1.0	1.0	1.0
Minimum Density				
Maximum	6.0	12.0	15.0	19.0
Mean	3.1	4.6	4.0	4.8
Median	3.0	4.0	3.0	3.0
Minimum	1.0	1.0	1.0	1.0

Table 6.4: Amount of backtracking using the parent-ancestor feed-through model

model, the parent-ancestor feed-through model was also tried to see if its effects would also be as local. Table 6.3 shows the percentage of partition overflows resolved and Table 6.4 shows the amount of backtracking. Using this model provides more ways of resolving overflows at each level in the hierarchy, but may involve propagating changes through large portions of the design independent of which level actually resolves the overflow. For example, when processing the subproblem containing the overflowed partition, the chosen ancestor partition might be the partition of the root region. Fortunately, Table 6.4 shows that of the overflows that are resolved, most of them are resolved quite locally. The median number of levels back in the hierarchy is 3 or 4. Table 6.3 shows that the use of a more general re-route model also improves the success rate considerably. The percentage of overflows resolved ranges from 68.9% to 85.7%.

Since these two models have exhibited local backtracking behavior, in future work, the more general re-route schemes described earlier will be investigated. In particular, the detour re-route for partition overflows does not a priori allow for any more propagation than the parent-ancestor feed-through model and thus may help increase the percentage of

overflows that may be resolved with out incurring any additional penalty. The ancestor-ancestor feed-through is also of some interest, but requires two initial pin reassignments and may require more propagation on average.

Other areas for future work include improving the selection heuristics for choosing which nets to try to re-route, trying less restrictive rules on pin reassignments to previous locations, and implementing strong modification within the context of hierarchical decomposition. Since strong modification implies a net-at-a-time process and one of the strengths of the hierarchical decomposition is its consideration of all nets simultaneously, any strong modification process most be contained within each level of the hierarchical decomposition.

These experiments show that a simple re-route technique can be effective on a hierarchy of subproblems. The hierarchical application of the rip-up and re-route is consistent with the view of hierarchical decomposition as a gradual refinement from rough approximation to final solution. The bottom-up rip-up and re-route confines the modifications to the smallest possible subtree at each level of the hierarchy. Thus, the decomposition is a gradual refinement and the rip-up and re-route is a gradual backtrack. Another way of describing the hierarchical rip-up and re-route is that it provides a continuum of weak modifications. This range of changes may be defined as *very weak* to *almost strong modifications.* A very weak modification would be when an overflow is resolved using the simple parent-grandparent feed-through model and an almost strong modification would be when an overflow is resolved by a pin reassignment on a partition near the top levels of the hierarchy. This last case is not a strong modification because the net is not removed from the problem for later re-insertion. However, the modification may require trying to re-route the whole net immediately and thus merits being called an almost strong modification.

Chapter 7

Conclusion

The use of hierarchical decomposition for general area routing has been investigated in this report. In particular, experiments were conducted to identify methods of handling the related subproblems of region decomposition, net decomposition, subproblem formulation and ordering, and rip-up and re-route. Though further work is necessary, the results of the current experiments are promising and informative. For each related subproblem, important characteristics of the corresponding solution method have been identified, and comparisons of different solution alternatives have been made.

The most detailed experiments in this report investigated ways of performing routing region decomposition. For this problem, it is important to account for both size and topological aspects of the routing region in order to reduce the complexity of subsequent subregions effectively. Based on the results of these experiments, a scan-line-based technique is proposed to generate optimal physical partitions of the routing regions. Such a technique provides a fast and simple method for applying the desired heuristics. The choice of partition is critical in that it distributes the complexity of the routing problem across the hierarchy. A priori, the more accurate information provided by irregular cut paths is advantageous, but the supporting heuristics and algorithms of the current experiments were not always able to exploit this information. In particular, though the capacity information of an irregular cut path is more accurate, it is still local in nature. Thus, at higher levels in the hierarchy, this information should not be used alone, rather it should be used to interpret, assign or supplement capacity information derived from more global observations. Also, the pin assignment heuristics must utilize the extra information represented by the topology of the cut path. A disadvantage of irregular cut paths is that more complex geo-

metric operations are required. This is most apparent in calculating net densities across a partition and in physically partitioning the geometry of a routing region. However, as the complexity of routing problems increase, the need for these complex operations will increase in general, and the cost of using irregular cut paths relative to the total cost or relative to other techniques will diminish.

The net decomposition problem is a critical aspect of hierarchical decomposition since the solutions to this problem most directly affect the complexity of subsequent routing subproblems in terms of net congestion. In this work, the pin assignment problem was formulated as a linear assignment problem. To properly handle irregular cut paths, either a new model or more complex cost functions will be required. The key problem is modeling the cost of assigning a net to a position relative to all the nets rather than the individual net. For irregular cut paths, this process is complicated by bends in the cut path that may represent blockages to some of the nets. Proximity cost functions seem the most effective for assigning pins to irregular cut paths because they apply the more detailed information of the cut path to net abstractions in a way that maintains the effectiveness and validity of the information. Another important issue is allowing deviations from the standard wiring model. Historically, these deviations have been allowed in the context of very detailed routing information. An important area of future research is to understand how to allow these deviations in the context of a routing hierarchy. The key issues are determining what information is necessary and/or sufficient to allow deviations at higher levels in the hierarchy, and understanding and modeling the effects of the wiring model deviations at these levels.

The final phase in the basic hierarchical decomposition method is subproblem solution. As seen in hierarchical routing on grid graphs, a decomposition method that produces trivial final subproblems is attractive. Unfortunately, the pattern routing method attempted in this work was hampered by inconsistent pin and pseudo-pin models. Alleviating this condition will make the method much more tractable. In general, for the pattern routing technique to work, the subproblems must be trivial routing problems and must be natural consequences of the decomposition process. The pattern router is useless if the decomposition produces cases that can not be handled. On the other hand, the implementation of the decomposition process may warrant stopping higher in the hierarchy and using more sophisticated detail routing algorithms to complete the problem. The amount and kind of analysis required at each level of the hierarchy may be different. Thus, in addition

to speed considerations, algorithmic considerations may also indicate that larger or more complex final subproblems are preferable. Subproblem dependence and the related problem of subproblem ordering is also important. More research is required in this area.

A hierarchical rip-up and re-route method has been described in this report. It works bottom-up through the hierarchy in a manner that is consistent with the original top-down decomposition. If the decomposition is viewed as gradual refinement, then the rip-up and re-route can be viewed as gradual backtrack. The re-route is very restrictive relative to a re-route on a flat design in that changes are restricted to changes between nodes in the hierarchy, yet even this subset has demonstrated significant decreases in the number of overflows. Also, some of the more general ways of performing rip-up and re-route on the hierarchy have not yet been implemented. The current results indicate that the rip-up and re-route process can be confined to fairly small subtrees in the hierarchy and that it is probable that this condition will also hold for the more general methods. Thus, the addition of these more general techniques should increase overall performance.

Hierarchical decomposition is a powerful technique for handling complex routing problems. The hierarchy provides a uniform and consistent structure for gathering, analyzing, and applying information. In this context, the hierarchy represents a unification of global and detail routing in that it provides a method that gradually refines an approximate solution into a final, detailed solution. The results of this report provide insight on how to manage and exploit the flow of information in this hierarchy for general area routing problems.

Bibliography

[AHU74] Alfred E. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[BM87] P. Birzele and P. Michel. Reduced design and manufacturing time by new semicustom standards and VLSI-design methodologies. In *Fast-Prototyping of VLSI*, pages 169–177. North-Holland, Amsterdam, Netherlands, 1987. Selection of papers from IFIP WG 10.5 Workshop in Grenoble, France during March 17–19, 1987.

[BP82] Michael Burstein and Richard Pelavin. Hierarchical channel router. Research Report RC 9715, IBM T.J. Watson Research Center, Yorktown Heights, New York, November 1982.

[BP83a] Michael Burstein and Richard Pelavin. Hierarchical channel router. In *Proceedings of the 20th Design Automation Conference*, 1983.

[BP83b] 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.

[Chr89] Wayne A. Christopher. Mariner — a sea of gates layout system. Master's thesis, University of California, Berkeley, 1989.

[CYK+87] Tzoyao Chan, Alex Yuen, Kurt Knorpp, Ming Hung, Peter Tsao, Mike Freie, Yen Chang, Rick Rasmussen, Alex Hui, and Patrick Yin. Advanced structured arrays combine high density memories with channel-free logic array. In *Proceedings of the IEEE Custom Integrated Circuit Conference*, pages 39–43, May 1987.

[DAK85] Wei-Ming Dai, Tetsuo Asano, and Ernest S. Kuh. Routing region definition and ordering scheme for building-block layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(3):189–197, July 1985.

[Deu76] D. N. Deutsch. A "DOGLEG" channel router. In *Proceedings of the 13th Design Automation Conference*, pages 425–433, June 1976.

[DK82] 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.

[HCBA82] C. W. Ho, D. A. Chance, C. H. Bajorek, and R. E. Acosta. The thin-film module as a high-performance semiconductor package. *IBM Journal of Research and Development*, 26(3):286–296, May 1982.

[HM85] Gary D. Hachtel and Christopher R. Morrison. Linear complexity dynamic programming algorithms for hierarchical routing. In *Proceedings of the International Symposium on Circuits and Systems*, pages 183–186, June 1985.

[HM89] Gary D. Hachtel and Christopher R. Morrison. Linear complexity algorithms for hierarchical routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(1):64–80, January 1989.

[HO84] G. Hamachi and J. Ousterhout. A switch-box router with obstacle avoidance. In *Proceedings of the 21th Design Automation Conference*, pages 173–179, June 1984.

[HPE+86] C. P. Hsu, R. A. Perry, S. C. Evans, J. Tang, and J. Y. Liu. Automatic layout of channelless gate array. In *Proceedings of the IEEE Custom Integrated Circuit Conference*, pages 281–284, May 1986.

[HS71] Akihiro Hashimoto and James Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proceedings of the 8th Design Automation Workshop*, pages 155–169, 1971.

[HWD+85] A. Hui, A. Wong, C. Dell'oca, D. Wong, and R. Szeto. A 4.1K gates double metal hcmos sea of gates array. In *Proceedings of the IEEE Custom Integrated Circuit Conference*, pages 15–17, May 1985.

[Joh83] D.S. Johnson. The NP-completeness column: An ongoing guide. *J. Algorithms*, 3:381–395, 1983.

[Joo85] R. Joobani. WEAVER: A knowledge-based routing expert. Research Report CMUCAD-85-56, Carnegie-Mellon University, June 1985.

[Lau87] Ulrich Ph. Lauther. Top down hierarchical global routing for channelless gate arrays based on linear assignment. In Carlo H. Séquin, editor, *Proceedings of the International Conference on Very Large Scale Integration*, Amsterdam, August 1987. IFIP TC, WG 10, Elsevier Science Publishers B.V.

[Law76] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[Lay88] 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.

[LTW86] W.K. Luk, D.T. Tang, and C.K. Wong. Hierarchical global wiring for custom chip design. In *Proceedings of the 23th Design Automation Conference*, pages 481–489, 1986.

[Luk85] W. Luk. A greedy switch-box router. *Integration*, 3:129–149, 1985.

[Meg79] Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, November 1979.

[MS84] 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.

[MS85] M. Marek-Sadowska. Two-dimensional router for double layer layout. In *Proceedings of the 22th Design Automation Conference*, pages 117–123, June 1985.

[MS86] Malgorzata Marek-Sadowska. Route planner for custom chip design. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 246–249, November 1986.

[Nis] Yoshihito Nishizaki. Region ordering of a hierarchical router for Sea-of-Gates. EECS 244 Term Project Report, Fall Semester, 1987, U.C. Berkeley.

[Ott82] R. H. Otten. Automatic floorplan design. In *Proceedings of the 19th Design Automation Conference*, pages 261–267, 1982.

[PSK85] Ash M. Patel, Norman L. Soong, and Robert K. Korn. Hierarchical VLSI routing – an approximate routing procedure. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(2):121–126, April 1985.

[RF82] Ronald L. Rivest and Charles M. Fiduccia. A "greedy" channel router. In *Proceedings of the 19th Design Automation Conference*, pages 418–424, 1982.

[RSVS85] James Reed, Alberto Sangiovanni-Vincentelli, and Mauro Santomauro. A new symbolic channel router: Yacr2. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(3):208–219, July 1985.

[SSV87] 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.

[TS88] 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.

[WM86] K. Winter and D.A. Mlynski. Hierarchical loose routing for gate arrays. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 238–241, November 1986.

[YK82] Takeshi Yoshimura and Ernest S. Kuh. Efficient algorithms for channel routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-1(1):25–35, January 1982.