1986

Intra region routing

Robert Alan Eustace
University of Central Florida

INTRA REGION ROUTING

by

Robert Alan Eustace

A dissertation submitted in partial fufillment of the requirements
for the degree of Doctor of Philosophy in
The Department of Computer Science at
the University of Central Florida
Orlando, Florida

August 1984

Major Professor: Amar Mukherjee

ABSTRACT

The custom integrated circuit routing problem normally requires partitioning into rectangular routing regions. Natural partitions usually result in regions that form both "channels" and "areas". This dissertation introduces several new channel and area routing algorithms and measures their performance.

A formal description of the channel routing problem is presented and a relationship is established between the selection of intervals for each track and the number of tracks in the completed channel. This relationship is used as an analysis tool that leads to the development of two new and highly effective channel routing algorithms: the Revised and LCP algorithms. The performance of these algorithms is compared against the Dogleg, Greedy, and several area routing algorithms over sets of randomly generated channels. The results indicate performance increases ranging from 2.74 to 34 times, depending on the characteristics of the channel.

In area routing, a new Degree of Freedom(DOF) based algorithm is developed that is straightforward to implement, is extensible to multipoint nets and reports if a path does not exist to complete the net. The quality of area routing algorithms is measured by the difficulty of the areas that can be successfully routed over sets of randomly generated areas. An extended definition of Manhattan Area Measure (MAM) is introduced as a measure of the difficulty of completing the wiring for areas with multipoint nets.

The results show that the DOF algorithm has higher completion rates than the Lee algorithm. This difference is greatest in areas with high aspect ratios. A new measure of the difficulty of an area is developed that places upper bounds on the performance of area routing algorithms. In areas with low aspect ratios, the drop in algorithm completion rates is closely related to this upper bound.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Amar Mukherjee, for his support and guidance in this work. It was his vision and perseverance that pushed us into the mainstream of VLSI research when only a handful of schools had such programs.

I would also like to thank the members of my committee: Dr. Charles Hughes, Dr. Robert Brigham, and Dr. Brian Petrasko.

I would also like to thank John Newkirk and Rob Mathews. The time I spent working with them at Stanford was an exceptional learning experience. I would also like to thank Tim Saxe for the numerous enlightening discussions on integrated circuit routing and Edwardo Diaz for his implementation of the Greedy channel routing algorithm. I would also like to thank Tim Curry, my office mate and confidant. He has remained a trusted friend through it all. I also would like to thank Eddie Haralson, Ben Goldfarb, Phil Caron and the operations staff for their support throughout the preparation of this dissertation.

Finally, I am deeply grateful to my family and friends for their understanding and support through the years.

This work was supported in part by NSF grant MCS-8005096A02.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

With the advent of Very Large Scale Integrated (VLSI) circuits, the number of devices that can be placed on a chip is quickly exceeding the ability of human designers to interconnect them. The interconnection of elements of the circuit is called *routing*. For large circuits, the routing can easily exceed 30% of the entire design time. This time leads designers to forego any late changes to the design, for fear of having to reconnect the new pieces. The design of adequate tools for integrated circuit design and routing is imperative if we intend to exploit the power of VLSI.

These circuits are usually created bottom up. A designer creates small cells that can be combined into larger cells. A cell that does not use lower level cells is called a *leaf cell*. Figure 1.1 gives an example of a leaf cell: a nMOS shift register. The designer might enter the cell using a color graphics terminal, with each color representing a different layer in the fabrication process. For example, a red

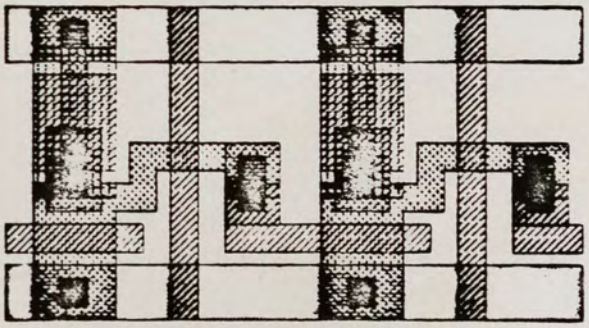


Figure 1.1 A nMOS shift register cell.

rectangle might correspond to a polysilicon area in the fabricated chip. These cells are then used as components in more complicated cells.

A composite cell is created by calling lower level cells and placing them in the area of the current cell. Abstractions of lower level cells are called *modules*.

This dissertation focuses on the interconnection of functional modules on Large Scale Integrated (LSI) and Very Large Scale Integrated (VLSI) circuits. An example of this problem is shown is Figure 1.2.

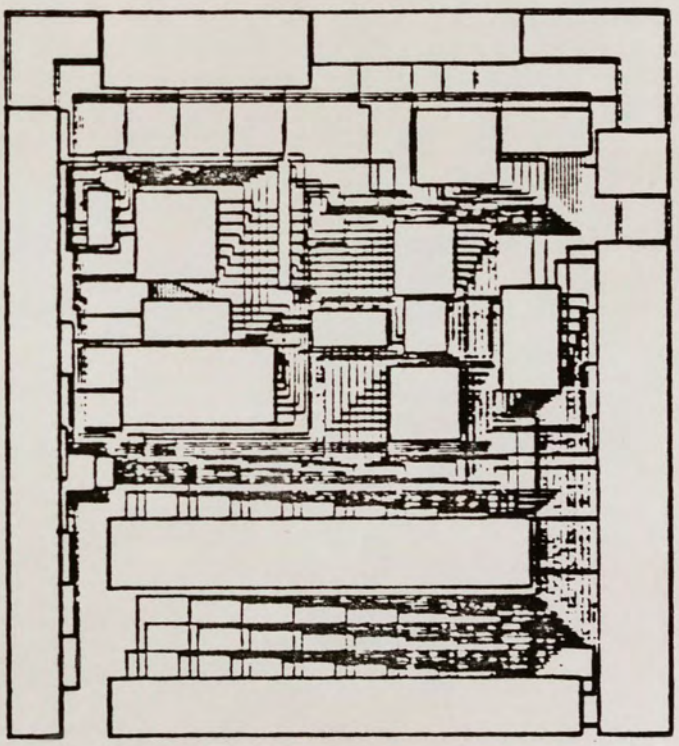


Figure 1.2. A Custom routing wiring solution.

At the edge of each module are points that the designer would like interconnected. These points are called *pins*. Pins are connected to other pins using wire.

The input to the general interconnection (also called general routing) problem consists of a set of *module specifications* and a set of *signal net definitions*. For our purposes, a *module* is specified by a rectangular bounding box with fixed dimensions and an associated set of pins. Each pin specifies a fixed location along the perimeter of a module, where a wire should be connected. Figure 1.3 shows an example of the general routing problem and a possible solution. A *signal net* is defined to be a set of pins which are to be electrically connected by wire. A *two point* signal net has only two pins in the set while a *multipoint* (or *multipin*) net has more than two pins. The nets are assumed to be composed of disjoint pins. The *routing region* is

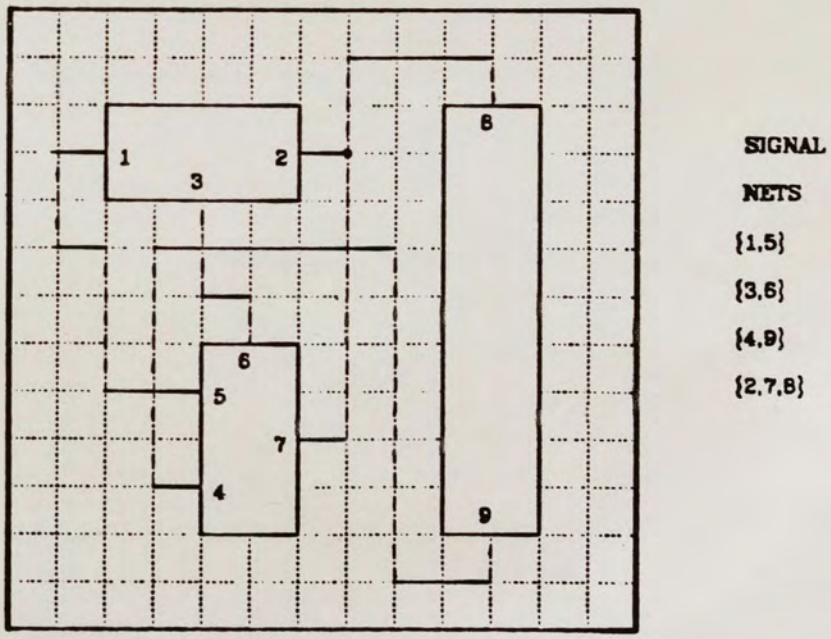


Figure 1.3. General routing problem.

defined to be the area available for routing the wires. The routing region is gridded, with all pins and wires aligned on this grid. We will assume there are only two layers available for wires, typically polysilicon and metal. The two layers can be electrically connected using a structure called a *via*. For reasons that will be explained in Chapter 3, we will assume that one layer always runs horizontally and the other always runs vertically. The net {2,7,8} is a multipoint net, while the others are two point nets. Note that wires are not permitted to pass over any module.

Layout organizations that allow the designer to specify both the size and placement of the modules are called *custom* layouts. *Gate arrays* are another popular layout organization. In gate arrays, the only leaf level cells are logic gates (such as 2 input NAND gates). The cells are organized as a fixed number of rows with a certain number of gates per row. The rows are always separated by a constant sized routing region called a *channel*. Figure 1.4 illustrates the gate array organization.

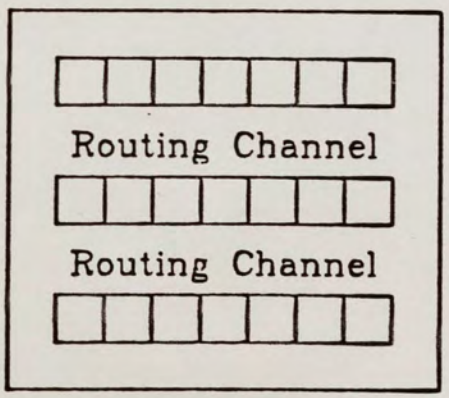


Figure 1.4. Gate array layout organization.

To implement a gate array style layout, a designer gives the gate array producers a circuit diagram. They, in turn, map gates in the circuit to gates on the chip and specify the wiring necessary to connect those gates on the chip. The advantage of gate arrays is that, for the most part, the translation from circuit diagrams to final wiring can be automated. Also gate arrays have speed and cost advantages (in high enough volumes) over conventional TTL logic.

The disadvantages are many. The fixed module size forces the designer to implement complex functions using very simple building blocks. It is similar to implementing an ALU using nothing but individual NAND gates. The fixed placement restricts the ability of the designer to group modules according to their function. Thus gate array organizations have a very poor density of active elements (transistors) per unit of area. This waste of active area limits chip complexity, increases cost (because fewer chips can be placed on a single wafer), decreases speed and increases power consumption.

To utilize the full power of integrated circuit technology, circuit designers are turning to Computer Aided Design (CAD). Here, computer scientists are becoming deeply involved in the design of powerful tools that will give some leverage toward solving the very difficult problems posed by VLSI. The interconnection problem is one of the more difficult of these challenges.

As chip size increases, the complexity of the routing problem grows too quickly to allow the entire problem to be routed as a single entity. For this reason, the routing region is usually divided in rectangular blocks, and each block is routed separately. The interconnection of pins along the perimeter of rectangular blocks is

called the *Intra region routing* problem, and the characteristics and algorithms that can be applied to this problem is the focus of this dissertation. If fixed pins are located on all four sides of the region, the problem is called the *switchbox* or *area* routing problem. If fixed pins are only located on the top and bottom edges, the problem is called the *channel routing* problem. Two very different algorithms are used to route each of these problems. The quality of a channel routing solution is measured by the number of horizontal rows in the completed channel. Each row in the solution is called a *track*. The number of tracks is compared against the *density*, the minimum number of tracks that must be used to complete the wiring. In area routing problems, the quality of the area routing algorithms is measured by comparing the number of problems attempted to the number of problems where the wiring is successfully completed. This ratio is called the *completion rate*.

Chapter 2 reviews past work in integrated circuit routing. Several algorithms that are described in this chapter are used in the comparisons presented in later chapters.

Chapter 3 analyzes channel routing algorithms. The discussion begins with a formal definition of the channel routing problem. This formalization allows the succinct description of each of the more popular channel routing algorithms. This formalization also allows a relationship to be established between the selection of wiring for the current track and the number of tracks in the final solution. This relationship is used to prove the optimality of the Left Edge channel routing algorithm. The relationship also allows the "diagnosis" of the conditions that led to the failure of the algorithm to complete a given channel in the density. A careful analysis of the effectiveness of

previous algorithms led to the development of two new channel routing algorithms that are shown in Chapter 5. Each significantly reduces the number of tracks necessary to complete the channel routing problem. The chapter ends with a discussion of the problems associated with using exclusively channel routers in routing custom integrated circuits.

Chapter 4 analyzes area routing algorithms. M.I.T.'s Placement and Interconnect project is discussed as an example of the use of area routing algorithms to route custom integrated circuits. The quality of area routing algorithm is measured by the difficulty of the areas that it can successfully route. The Manhattan Area Measure (MAM) is used as a measure of the difficulty of completing the wiring for areas with two point nets. This chapter also examines the sequential nature of area routing algorithms. These algorithms typically route one wire at a time. Algorithms with this characteristic must use caution to route each wire in a way that minimizes the effect of the path on nets yet to be routed. Three important considerations are layer assignment, pin blocking and vias. Minimizing the number of vias is of special importance in achieving high completion rates. This section uses *systems of distinct representatives* to analyze why routing simplest paths first is a good strategy for minimizing the number of vias in the solution. A new Degree Of Freedom (DOF) based algorithm is developed that is functionally equivalent to the LUZ router, a special purpose area router for two interconnecting areas with two point nets. The DOF algorithm is easier to implement, is extensible to multipoint nets and reports if a path does not exist to complete the net. To measure the difficulty of areas with multipoint nets, this chapter extends the MAM to multipoint nets. This extension is based on

developing a lower bound on the rectilinear Steiner tree. This bound is shown to be exact for two and three point nets.

Chapter 5 compares area and channel routing algorithms over a wide variety of sample channels. Sample channels are generated using algorithms developed at M.I.T. and by the author. The area routing and channel routing algorithms are extended to allow reasonable comparisons. The results clearly show that the area routing algorithms take more tracks to route the same channels. Of the channel routing algorithms, the two new algorithms developed in Chapter 3 give very close to optimal performance, exceeding the performance of well known channel routing algorithms by between 2.74 and 20 times, with an average improvement of 10.

Chapter 6 compares several area routing algorithms over randomly generated test cases. These statistics show an improvement in completion rates when using the DOF algorithm as opposed to other popular area routing algorithms. The results show that these improvements occur for both two point and multipoint nets using areas with several different aspect ratios. The results also show a dramatic decrease in completion rates as the complexity of the region reaches a certain threshold. This chapter develops a measure of the difficulty of the routing regions that is used to show upper bounds on area routing algorithms performance. The statistics generated in this section can also be used to determine how large the routing region must be to give the routing algorithms a reasonable chance of completing the wiring.

The final chapter concludes the discussion of intra region routing and examines some questions remaining for future research.

CHAPTER 2

Previous Work

This chapter reviews previous work concerning integrated circuit routing. The algorithms discussed are the Lee Algorithm, the Hightower algorithm, the LUZ algorithm and channel routing algorithms.

2.1. The Lee Algorithm

The oldest and still the most popular method of routing custom integrated circuits is the Lee algorithm[Lee61] (henceforth the Lee). The Lee is a maze search algorithm that is guaranteed to find the shortest path, if one exists. The algorithm has been widely used in both printed circuit board and integrated circuit routing.

The Lee begins with an arbitrary shaped gridded routing region with any number of obstacles. Grid points are called cells. Two pins are labeled as the source and target. The Lee labels each empty adjacent cell to the source with a cost 1, the cost of reaching that cell from the source. Since wires cannot be placed diagonally, these cells are not considered adjacent to the source. Then the Lee identifies each empty cell that is adjacent to cells with a cost one and labels these with a cost two. The edge of the labeled cells is called the *wavefront*. This process continues until either the target pin has been labeled or there are no remaining empty cells adjacent to any labeled cells. In the latter case, the Lee reports that there is no path from the source to the target.

If the target pin has been labeled, the algorithm *backtracks* from the target, moving at each step to any cell with a lower cost. Figure 2.1 shows the propagation pattern of the Lee search and a possible backtrack path. Here, the pins have not been placed on a module boundary to highlight the diamond shaped wavefront propagation of the Lee.

Another way of looking at the expanding wavefront is to consider it as a breadth first search[Mo57], from the source to the target, where all edges (as paths between cells) have unit cost.

The Lee can also be extended to create different costs for each routing layer and for vias. These costs allow the algorithm to establish preferential directions for each routing layer, penalties for crossings and additional costs for vias. The use of variable costs adds a great deal of flexibility to the Lee.

The extended Lee uses a simple graphical model of the routing region. Each cell is replaced by a pair of nodes, one for the lower

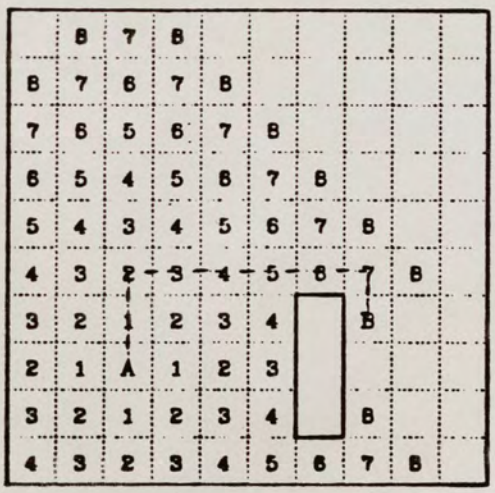
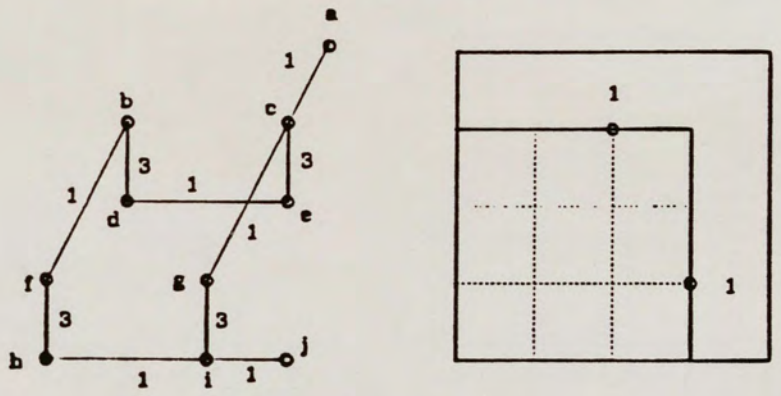


Figure 2.1. Lee algorithm wavefront expansion.

layer and one for the upper layer. Edges form the possible path directions into and out of each node. For example, if we restrict the upper layer to the vertical direction and the lower layer to the horizontal direction, Figure 2.2(a) illustrates the graphical model for a simple example.

Using this model, costs can be assigned to edges in the graph. In this example, a cost 1 is placed on all horizontal and vertical edges and a cost 3 on the via edges. This might cause the algorithm to choose a path that has more horizontal and vertical edges, but less vias. The importance of variable costs is explained in Chapter 4.

The Lee is modified by replacing the breath first search with a general single source shortest path algorithm. The one used in our



(a)

Step	N	Distance list
Initial	{a}	D(c)=1
1	{a,c}	D(g)=2, D(c)=4
2	{a,c,g}	D(e)=4, D(i)=5
3	{a,c,g,e}	D(i)=5, D(j)=6
4	{a,c,g,e,i}	D(d)=5, D(j)=6, D(h)=6
5	{a,c,g,e,i,d}	D(j)=6, D(h)=6, D(b)=8
6	{a,d,g,e,i,d,j}	path found!

(b)

Figure 2.2. Graph model and Dijkstra's algorithm.

implementation of the variable cost Lee is Dijkstra's algorithm[Di59].

Dijkstra's algorithm is a step by step procedure where, by the kth step, the shortest paths to the k nodes closest to the source have been calculated. These nodes are contained in the set N. At the $k+1^{th}$ step, a new node is added to N whose cost to the source is the shortest of the remaining nodes outside of N. For example, in Figure 2.2(b), node a is selected as the source. At the initial step, node a is placed in the set N and the cost to each adjacent node is computed. In the first step, we add node c to the set N and recompute the costs using node c's adjacent nodes. If any of the recomputed costs form a cheaper path than was previously found, the old distance is replaced in the cost list. This process is repeated until the target node has been added to the set N. Simple bookkeeping allows the path to be backtracked to the source. The path found in the example is (a,c,g,i,j).

One of the disadvantages of the Lee is its memory requirements. Each grid point is required to store the cost of reaching the cell, along with some additional information regarding the type of object located at that cell. In large designs, the total could easily exceed one million grid points.

Akers[Ak67] suggested improvements in the cost labeling at each cell, reducing the number of bits necessary at each grid point to just two. The first expansion of the wavefront is done by labeling each cell adjacent to the source with a one. The second expansion labels each adjacent cell, also with a one. The next expansion labels adjacent cells with a two. The following expansion also uses a two. The process then

repeats, with the next expansion using a one, until the target is reached or there are no longer any more cells in the wavefront.

The backtrack procedure is changed to backtrack cells with the pattern ...1,1,2,2,1,1.. etc., starting with the number labeled in the target cell. In most current implementations, the "one-two" labeling of Akers is replaced by a backward pointer to the previous cell in the path. The backtracking routine simply follows the pointers back from the target to the source.

The second disadvantage of the Lee is its execution time. Figure 2.1 reveals that the Lee algorithm extends its wavefront in all directions at equal rates. Rubin[Ru74] suggested that since the target is known, we should expand more in that direction, unless that direction is blocked. His router achieves this by sorting the wavefront using a key which is a total of the true distance from the source plus the manhattan distance to the target. This technique places a penalty on moves away from the target. The technique also preserves the characteristic that the path found will always be the shortest. Soukup[So78] developed the arrow router which achieves a similar effect by controlling the loading and unloading of the stack containing the wavefront entries.

If we are willing to sacrifice the guarantee of minimality, Korn[Ko82] suggests techniques that will *overpull* the path toward the target. This technique is fundamentally the same as Rubin's, except that the penalties placed on moves away from the target are increased. This tends to "pull" the wavefront in the direction of the target. His statistics indicate order of magnitude performance increases over the traditional Lee.

2.2. The Hightower Algorithm

David Hightower[Hi69] suggested an algorithm that produces routing completion rates similar to the Lee but reduces execution time and memory requirements. The fundamental difference between this router and the Lee is that it does not store the entire plane in a matrix. Instead, only lines and points (as zero length lines) are stored. When finding a path between two points, it is necessary to test only those lines that currently exist.

The Hightower algorithm begins by constructing vertical and horizontal expansion lines emanating from the source and target terminals. Then, for each line, it finds the longest perpendicular escape line. If there is a multiple choice, the escape line nearest to the starting terminal is taken. This process is repeated until lines from both sets intersect, generating a connection. Figure 2.3 illustrates the Hightower expansion process. Once the two sets intersect, a backtracking procedure is used to make the connection

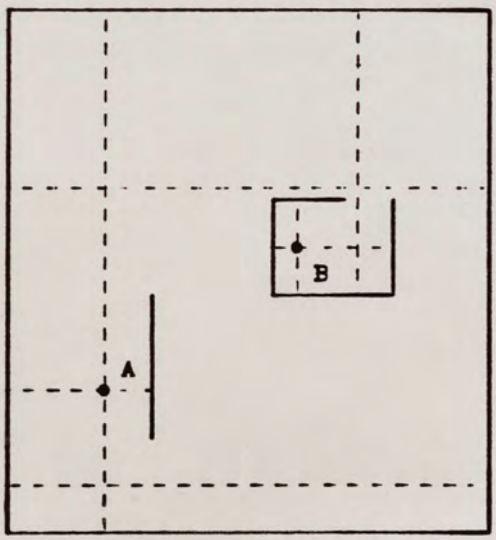


Figure 2.3. The Hightower expansion process.

between the two points.

The Hightower performs well when the routing area is small or has only a few connections. Its run time characteristics are poor on large, complicated mazes since these require the algorithm to keep track of a large stack of data. In these cases, the Hightower does not guarantee a connection, even if one exists.

The Hightower algorithm's insistence on choosing the longest perpendicular escape line can cause poor paths to be selected. Consider Figure 2.4. Here the path selected will be much longer and contain more vias than is necessary. Hightower reports only a 10% increase in path length using his algorithm although the results were reported for printed circuit boards where module obstructions are not present. The performance of this approach on the custom routing problem is unknown.

One interesting feature of the Hightower is that it tends to use paths with the minimum number of vias. The importance of this feature is detailed in chapter 4 and may be partly responsible for the

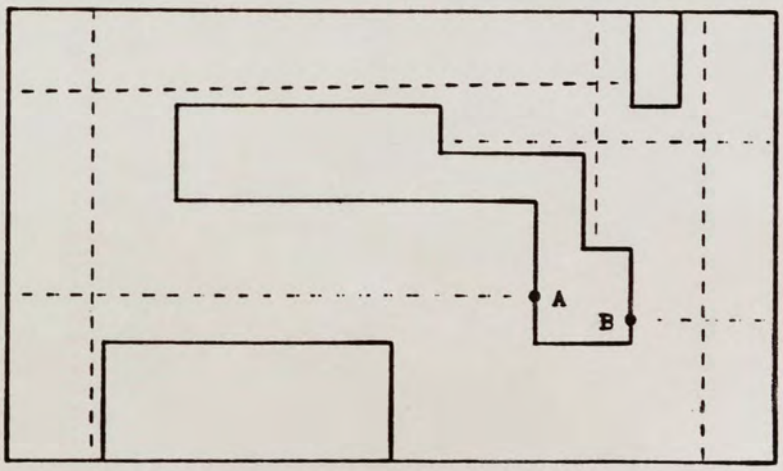


Figure 2.4. Poor routing using the Hightower.

Hightower's completion rates being similar to the Lee's, even though the Lee always finds a connection, if one exists.

Two algorithms have been developed that attempt to merge the advantages of the Lee with the line searches performed in the Hightower. Mikami[MiTa] developed an algorithm that is identical to the Lee, except that, rather than simply marking the adjacent cells with a cost, the algorithm marks all the costs along the horizontal and vertical expansion lines of the Hightower. The Lee will expand the lowest cost entry in the region, again extending the the costs as in the Hightower. Mikami reported a speedup of 2 over the unmodified Lee. Moreover, his approach can guarantee that it will find the shortest path connection, if one exists.

Hayns[Ha80] developed an algorithm, also based on the Lee, that expands similar to the Mikami approach, except that only the line segments bounding the wavefront are stored rather than the whole grid.

2.3. The LUZ Algorithm

The LUZ algorithm[Sm83] was developed by Lyle Smith and Tim Saxe at Stanford University as a means of analyzing path complexity. The algorithm starts with a rectangular routing region with pins on all four sides and no obstructions. This is commonly called the *area routing problem* or the *switchbox problem*. Because the LUZ is primarily an analysis tool, it deals only with two point nets.

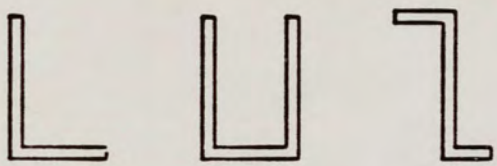
2.3.1. Path Shapes

The LUZ gets its name from the three basic path shapes it uses: L shaped paths, U shaped paths and Z shaped paths. These paths are

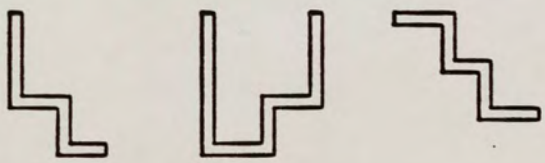
called LUZ level 1 paths. Assuming there is no other wiring in the region, any two point net can be wired using one of these basic path shapes. LUZ level 2 paths are formed by adding two L shaped corners into a LUZ 1 path. Higher LUZ number paths can be similarly defined. Figure 2.5 illustrates LUZ level 1 and some typical LUZ 2 paths.

2.3.2. The LUZ Algorithm

The LUZ attempts to minimize the number of vias by routing simplest paths first. Thus, the algorithm attempts to route each net using a LUZ level one path. In statistical studies using hundreds of randomly generated examples, Smith showed 75% of the nets can be routed using level one paths. The remaining nets are tried using LUZ level 2 paths. These studies showed that another 20% of the nets can be routed using these path shapes. The analysis goes on to show that almost no improvement comes from routing paths at a complexity higher than LUZ level 3. Unfortunately, the LUZ algorithm does not



(a) LUZ level 1 paths.



(b) Examples of LUZ level 2 paths.

Figure 2.5. The LUZ path shapes.

have any means to determine when to halt. It simply believes it has not tried a complex enough path.

Smith compared the results from the LUZ algorithm with the Lee for the same examples. The results showed that the LUZ using only LUZ level 2 paths had completion rates equal to the Lee. The LUZ exceeded the Lee in completion rates if level 3 paths were used. Figure 2.6 shows a very dense example routed using the LUZ algorithm.

Smith contends LUZ level 3 paths are very difficult to algorithmicly construct and concluded that a composite algorithm combining the LUZ and the Lee would produce excellent results. The

Figure 2.6. A dense routing example.

composite algorithm would have the LUZ route level 1 and level 2 paths. If any nets remain unrouted, the Lee would be used to complete the remaining nets. Since the Lee algorithm's execution time is proportional to the sparsity of the routing region, the Lee should run very quickly if 95% of the routing has been completed.

Chapter 4 redefines the LUZ algorithm, explaining the performance improvements and improving the definition of the simplest path. The new implementation has the advantage that it uses the Lee algorithm, realizes when to halt, and can be extended to multipoint nets.

2.4. Channel Routing

Channel routers have received a great deal of attention in the literature because of their applicability to gate array layout organizations.

The channel routing problem is given a rectangular region, called a *channel*, with pins located on the top and bottom edges. The region is gridded, with all pins and wires aligned on the grid. Horizontal grid lines are referred to as *tracks*. The objective of channel routing is to electrically connect all the nets using a minimum number of tracks. Figure 2.7 shows an example of a channel routing problem and a typical solution.

The horizontal segment of a net is determined by its leftmost and rightmost terminal connections. Let the local density, d_j, be the number of nets whose horizontal segments intersect column j. Since horizontal segments of distinct nets must not overlap, the maximum value of d_j will be a lower bound on the number of horizontal tracks

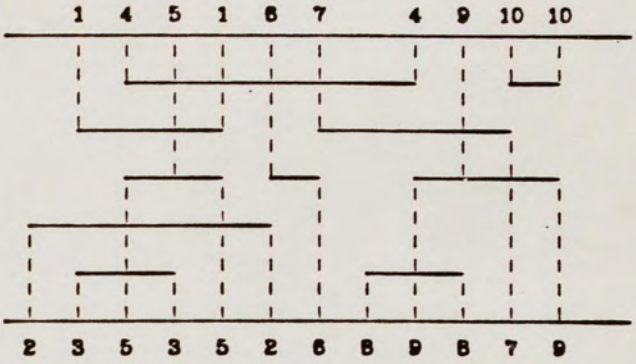


Figure 2.7. Channel routing example

necessary to route the channel. This lower bound is called the *channel density*.

The relationship between horizontal segments can be defined using an *interval graph*. Figure 2.8(a) illustrates the interval graph for the net list in Figure 2.7. Let $HG(V_{hg},E_{hg})$ be a interval graph where a node $v_i \in V$ represents net i and an edge $(v_i,v_j) \in E_{hg}$ if the horizontal segments of net i and net j cross a common column. This graph is commonly referred to as the *horizontal constraint graph*. In terms of the interval graph, the maximum clique number is the density.

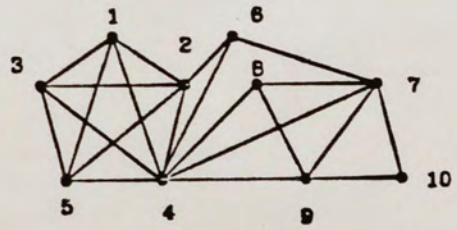


Figure 2.8. Horizontal constraint graph.

The "left edge" channel router of Hashimoto and Stevens[HaSt71] attempts to maximize the packing of horizontal segments in each track, starting from the bottom track. The method sorts the edges by the left endpoint of each segment. The sorted list for Figure 2.7 is (2,1,3,4,5,6,7,8,9,10). The algorithm selects the first edge, 2, and places it in the lower left corner of the routing region. Net 2 is deleted from the sorted list. It then scans through the remaining list for the first net that does not overlap net 2. In Figure 2.7, this would be net 7. This process is repeated until no more elements can be placed on track 1. The algorithm starts again using the remaining unplaced nets in the list and filling track 2. The track selections made by the left edge algorithm are shown in Figure 2.9.

Although the track assignments of the left edge appear to be reasonable, a problem arises when connecting the horizontal segments to the corresponding pins. Notice that to connect net 1 to the top of the channel, net 1 crosses directly over the end of net 3. Since net 3 will have to place a via at that position, this organization will short nets 1 and 3. Thus, net 1 must be above net 3. This was not taken into

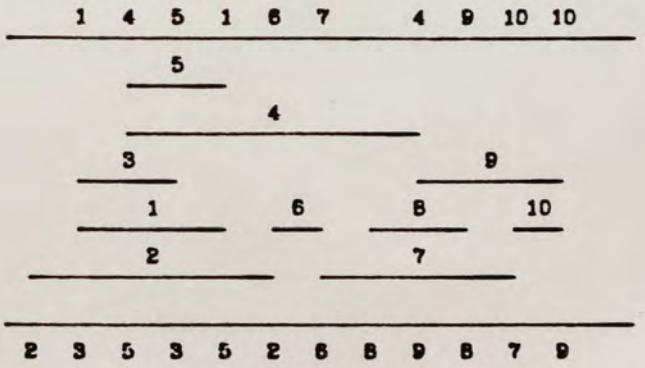


Figure 2.9. The left edge algorithm's track assignments.

account by the left edge algorithm. Constraints such as these are called *vertical constraints*.

To capture the vertical constraints we construct a *vertical constraint graph*. Let $VG(V_{vg},E_{vg})$ be a directed graph where $V_{vg}=V_{hg}$. Let $(v_i,v_j) \in E_{vg}$ *iff* there is a column containing a pin of net i on the top and one of net j on the bottom. An edge between nodes v_i and v_j indicates that the horizontal segment of net n_i must be placed above the horizontal segment of net n_j. The vertical constraint graph for Figure 2.7 is shown in Figure 2.10.

The "left edge" algorithm is only guaranteed to produce correct results if $E_{vg}=\phi$, indicating that there are no vertical constraints. This is the case if only pins of the same net are in the same vertical column. Chapter 3 shows that the "left edge" algorithm is guaranteed to find the optimal solution if there are no vertical constraints.

A "constrained left edge" algorithm was implemented by Persky.[PcDeSc76] This algorithm again places horizontal segments from the lower left corner of the routing region. The difference is that this algorithm will not place a horizontal segment for a net that has any descendants in the vertical constraint graph. Using the same example

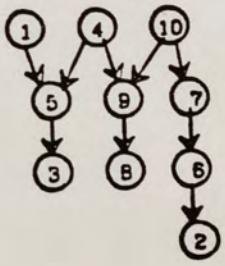


Figure 2.10. Vertical constraint graph.

as in Figure 2.9, net 2 can be placed since it is a leaf cell in the vertical constraint graph. Once a net has been placed, the corresponding node is deleted from the vertical constraint graph. The algorithm scans the remaining list of horizontal segments (sorted by left endpoint) for the first edge that does not overlap the previous net and has no descendants in the vertical constraint graph. Net 8 is the next net that fulfills both these requirements. This process is repeated, filling each track from left to right until all horizontal segments have been placed. Figure 2.11 illustrates the completed routing. In this case, the constrained left edge algorithm produced an optimal solution, since the number of tracks used is equal to the channel density.

The previous algorithm has disastrous results if there is a cycle in the vertical constraint graph. Cycles in the vertical constraint graph are called *vertical constraint loops*. Consider the channel routing problem and vertical constraint graph shown in Figure 2.12(a) and (b). In this example, the constrained left edge algorithm cannot route

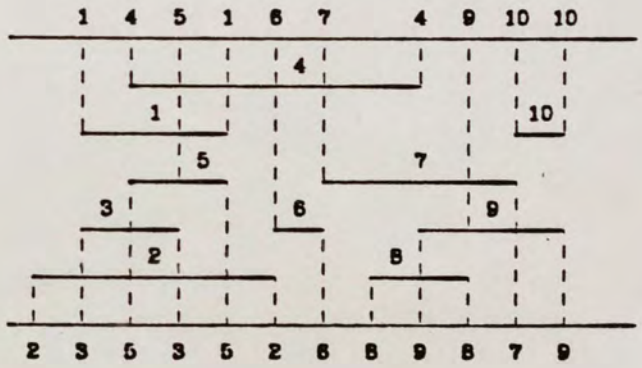


Figure 2.11. Constrained left edge track assignments.

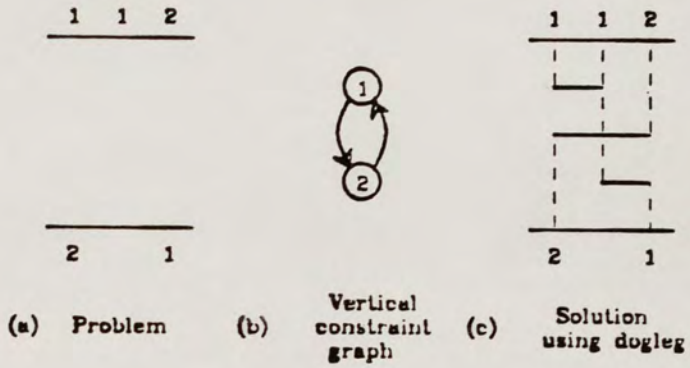


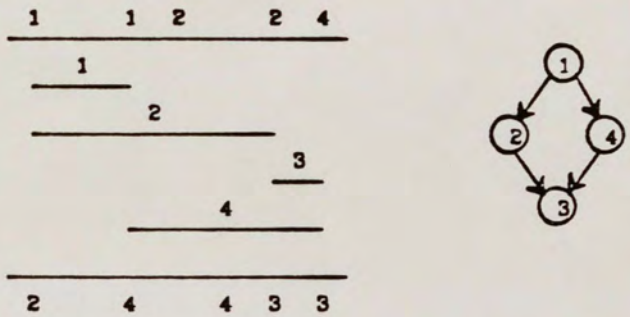
Figure 2.12. Vertical constraint loop.

either net 1 or net 2 since both have descendants in the vertical constraint graph. Figure 2.12(c) shows that a solution to this problem is only possible if we allow the horizontal segment of a net to be split. Splitting nets in this fashion is called *doglegging* [De76]. The *restricted* channel routing problem does not allow doglegging. The *Dogleg* algorithm proposed by Deutsch uses doglegging both to avoid vertical constraint loops and to decrease the density of the channel. The statement of Deutsch's algorithm given below has been modified based on an understanding of the vertical constraint graph. The routings produced by the revised algorithm are identical to the original, but the statement of the algorithm has been significantly simplified.

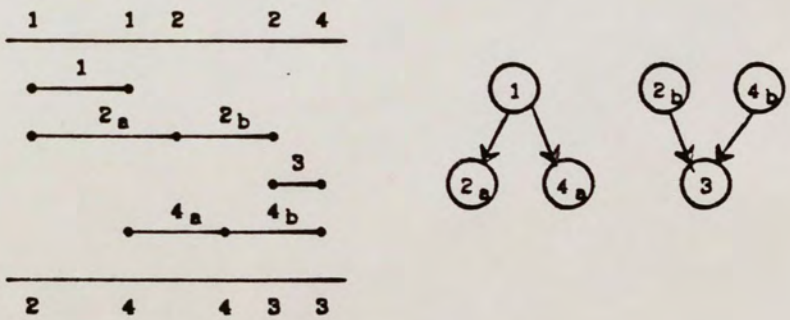
Deutsch's algorithm (henceforth the Dogleg) takes each multiple pin net and breaks it up into individual horizontal segments. A break occurs only in columns that contain a pin for that net. Figure 2.13 illustrates the horizontal segment definition and the corresponding

vertical constraint graph for both the "restricted" and the "unrestricted" algorithms.

A basic form of the Dogleg that is very similar to the Left edge algorithm is described below. Horizontal segments are sorted in increasing order of their left endpoint. The first segment in the list that has no descendants in the vertical constraint graph is placed in the channel. The node corresponding to this section of the net is removed from the vertical constraint graph. Then, the next net in the list that does not overlap the first segment and has no descendants in the vertical constraint graph is placed. This process continues for



(a) Restricted (no dogleg)



(b) Unrestricted (using dogleg)

Figure 2.13. Horizontal segment definition.

each track, from left to right, until all segments have been completed. Figure 2.14 illustrates the completed channel. Notice that the channel is optimal while the "restricted" channel router would require 4 horizontal tracks.

The Dogleg algorithm as suggested by Deutsch alternates between the filling of top and bottom tracks and fills bottom tracks from left to right and top tracks from right to left. A formal description of this algorithm is given in Chapter 3. Deutsch claimed that this symmetric alternating format produces routing with a smaller total vertical length than routing all nets from the bottom of the channel to the top.

Channel routers discussed thus far are only applicable on regions that have fixed pins on opposite sides of the routing region. Channel routers can be easily extended to connect to pins on the left and right edges of the channel, but the position of the connection must be left to the discretion of the algorithm. These pins are called *floating* pins, since the designer can only specify the side they connect to. Pins on the left and right edges of the channel are called *endpins*. Figure 2.15

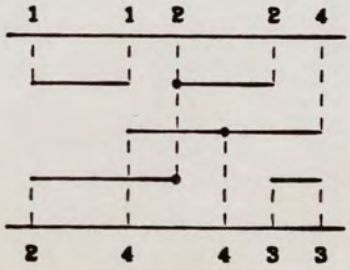


Figure 2.14. Dogleg channel routing example.

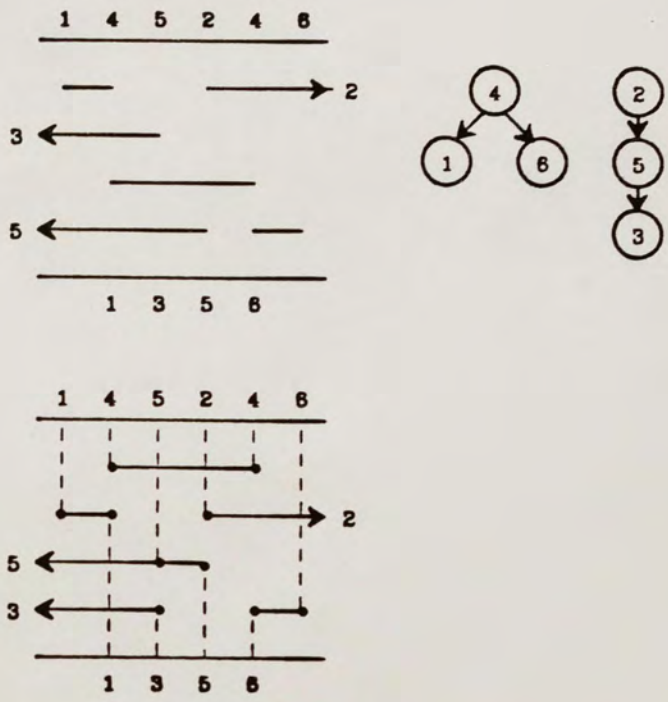


Figure 2.15. Channel routing example with endpins.

shows an example of a channel routing problem with endpins and the solution produced by the constrained left edge algorithm.

The Dogleg algorithm considers only a few of the possible candidates for merging. In Figure 2.16, for example, the Dogleg would consider only nets 2, 3 and 8 for placement on the first row, although other nets, such as 2 and 9, could also share a track, since they do not overlap and there is no directed path between them in the vertical constraint graph. Two algorithms by Yoshimura and Kuh[YoKu82] attempt to exploit alternate pairings of horizontal segments. The first algorithm attempts to minimize the longest path in the vertical constraint graph by attempting to combine those tracks that minimize the path through the vertical constraint graph. The second

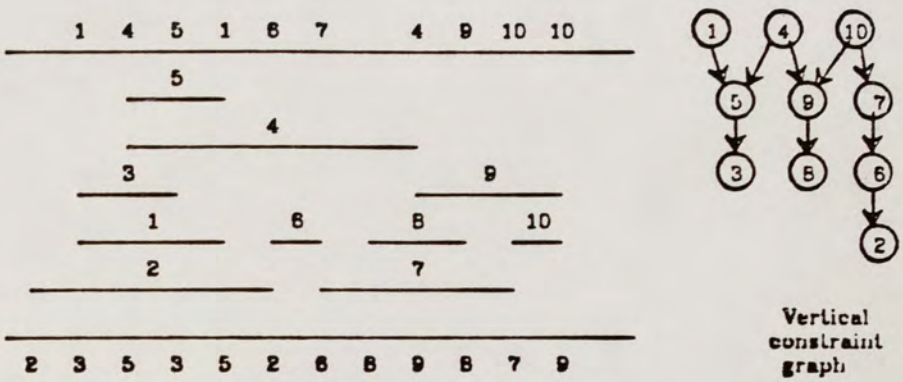


Figure 2.16. Merging of nets example.

algorithm achieves longest path minimization through matching techniques on a bipartite graph. Both techniques report better results than the Dogleg.

Both the Dogleg and the Yoshimura and Kuh algorithms will not work if there are cycles remaining in the vertical constraint graph.

The *Greedy* channel router[RiFi82, RiBaMi81] takes a totally different approach to the problem. The channel routing is done on a column by column basis (rather than row by row) beginning at the left side of the channel. In each column, the router tries to maximize the number of tracks available in the next vertical column using a sequence of heuristics.

Assume that the greedy algorithm has completed $i-1$ columns. A set of nets enters column i from the left. The Greedy algorithm constructs the wiring in column i using the following steps:

(1) Extend vertical segments from each terminal to either the first empty track or a track occupied by the same net, whichever uses the least vertical wire. Figure 2.17(a) shows an example. If no

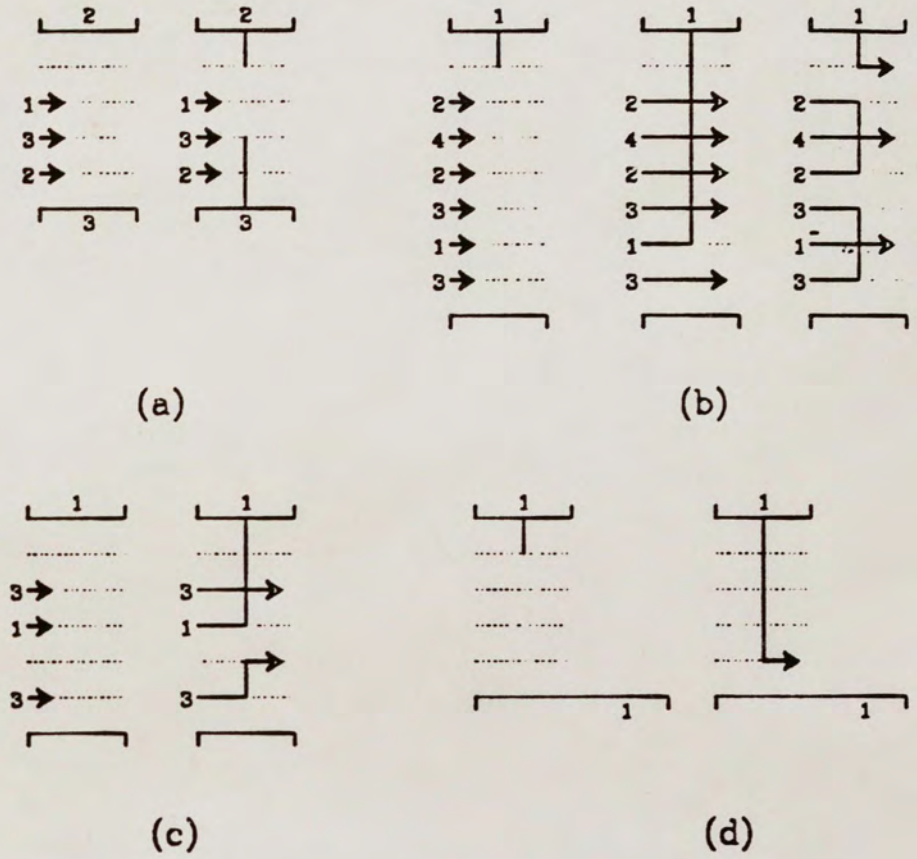


Figure 2.17. Greedy router example

such track is available, the Greedy router creates an additional track in the center of the channel.

(2) Choose vertical wiring that maximizes the number of empty tracks in column $i+1$. In Figure 2.17(b), the possible choices for vertical wiring are shown. Assuming that net 2 has no terminals to the right of column i and net 3 has a terminal to the right of column i, the second choice of vertical wiring will free 3 tracks and the first choice will free only 2 tracks. If there is a tie, the set of vertical segments that free tracks closest to the edges of the channel are chosen.

(3) Add vertical wiring to minimize the distance between tracks containing the same net. In Figure 2.17(c), a small amount of vertical wiring is added to reduce the distance between the two portions of net 3.

(4) Add vertical wiring to "pull" nets to tracks closest to the next terminal with that net number. In Figure 2.17(d), vertical wiring is added to bring net 1 to the bottom of the channel.

The performance and quality of the routing solutions generated will be measured in chapter 5. Since the approach does not use the vertical constraint graph, it avoids the vertical constraint loop problem.

A new algorithm by Burstein and Pelavin called the Hierarchical Channel Router[BuPe83] will handle these constraint loops but has better electrical characteristics than the "greedy" approach. This algorithm hierarchically decomposes the channel into cells, and then applies graph techniques to minimize the wiring within each cell. This approach was able to solve the most difficult example in the literature, Deutsch's example, in the optimal 19 tracks. The Dogleg was only able to solve the problem in 21 tracks. The Yoshimura and Kuh algorithms required 21 and 20 tracks respectively, but used considerably less vias than the Dogleg. The "greedy" algorithm also required 20 tracks, but required extra wire and vias.

CHAPTER 3

Analysis of Channel Routing Algorithms

Finding optimal solutions to the routing problem is very hard. LaPaugh[La80] showed that even the restricted channel routing problem (no doglegging) is NP-complete, and W. Donath [YoKu82] showed the unrestricted channel routing problem is also NP-complete. Since finding an optimal solution is very difficult, routing algorithms rely on heuristics to provide acceptable solutions.

Channel routers have three important characteristics:

(1) Efficient algorithms with straightforward implementations are well known.

(2) An algorithm's effectiveness can be measured against easily computed lower bounds. These lower bounds form a basis for comparing different channel routing heuristics.

(3) Channel routers can guarantee 100% completion rates if constraints are noncyclic. This guarantee is obtainable because channel routing algorithms can always, if necessary, add additional tracks in the center of the routing region without disturbing tracks that have been already placed.

In this chapter, we develop a formalization of the channel routing problem and prove a necessary and sufficient condition for a channel to be completed in the optimal number of tracks.

This condition will then serve

(1) as the basis for proofs on the optimality of channel routing algorithms.

(2) as an analysis tool to determine at what point in the routing a "mistake" occurred.

(3) as the basis for new channel routing heuristics that improve the number of tracks above the density necessary to complete the routing. The effectiveness of these heuristics will be measured in Chapter 5. The results indicate an improvement in algorithm performance ranging from 2.74 to 20, depending on the characteristics of the channel, with an average improvement of 10.

3.1. Formalizing the Channel Routing Problem

The channel routing problem can be formalized as the following decision problem. A simple example to clarify this notion is presented later in Figure 3.1.

Given: A set of terminals T and nets N. Each terminal, t, is an ordered pair (c,s) where $1 \le c \le n$ is the column and $s \in \{bot, top\}$ is the side. A terminal on the top or bottom of the channel is also called a *pin*. The set of terminals may also contain *endpins* of the form $(0, left)$ and $(n+1, right)$. Endpins are terminals connected to nets that enter or exit the left or right side of the channel. For $t=(c,s)$, let the function $C(t)=c$.

Let N be a collection of m disjoint sets of pins and possibly non-disjoint endpins. We let N_i, for $1 \le i \le m$, represent the i^{th}

net, such that $N_i=\{t_1,t_2,\ldots,t_l\}$ where $l\geq 2$ and t_i's are ordered such that $C(t_i)\leq C(t_{i+1})$ for $1\leq i\leq l-1$.

Let the *interval*, $p=[a,z]^i$, denote the set of all columns in net N_i between columns a and z inclusive. Two intervals, $[a,z]^i$ and $[a',z']^j$, are *incident* if $\{a,z\}\cap\{a',z'\}\neq\phi$. Thus, two intervals of different nets are incident if they share a common endpoint. The *interval set* P_i for net N_i is $P_i=\{p \mid p=[C(t_j),C(t_{j+1})]^i$ for $1\leq j\leq l-1$ where $t_j,t_{j+1}\in N_i$ and $C(t_j)\neq C(t_{j+1})\}$. Let $S=\{\bigcup P_i$ for $1\leq i\leq m\}$ be the set of all intervals.

Vertical constraints are defined by the directed graph $VG(V_{vg},E_{vg})$ where

$$V_{vg}=\{v \mid v\in S\}$$

$$E_{vg}=\{(v_1,v_2) \mid v_1\in P_i \text{ and } v_2\in P_j \text{ for } 1\leq i,j\leq m \text{ where } i\neq j \text{ and } v_1 \text{ and } v_2 \text{ are incident in column } x \text{ and } (x,top)\in N_i\}.$$

Horizontal constraints are defined by the undirected graph $HG(V_{hg},E_{hg})$ where

$$E_{hg}=\{(p,q) \mid p\in P_i \text{ and } q\in P_j \text{ for } 1\leq i,j\leq m \text{ where } i\neq j \text{ and } p\cap q\neq\phi.$$

Channel Routing Problem:

> Given a positive integer k, determine a mapping, δ, which assigns to every interval, $p \in S$, a number between 1 and k inclusive such that
>
> (i) if $(p,q) \in E_{vg}$ then $\delta(p) > \delta(q)$
> (ii) if $(p,q) \in E_{hg}$ then $\delta(p) \neq \delta(q)$

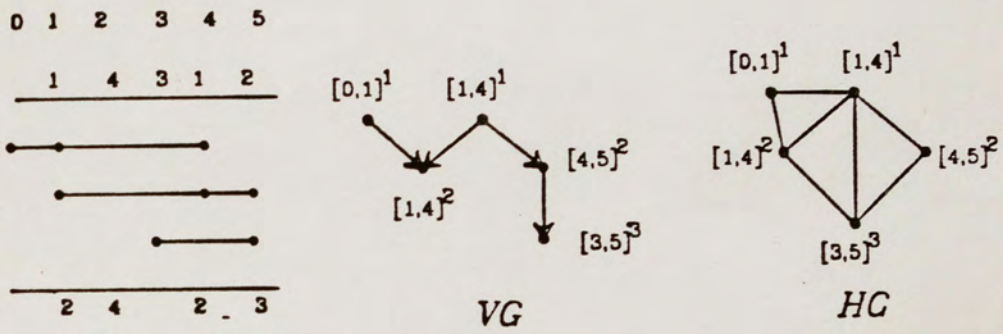
If such a δ exists, δ is called the channel assignment mapping and each integer in the range 1 to k represents a *track* in the channel.

Condition (i) represents vertical constraints, and condition (ii) guarantees that the horizontal segments of different nets do not overlap. Figure 3.1 gives a simple example using the notation described above.

The *density* for a set of intervals S' where $S' \subseteq S$ is defined as follows. For each net, N_i, let $Q_i = \{l \mid l \in \bigcup [a,z]^i \in S'\}$ represent the set of all columns defined by the intervals in S' that belong to net i. Let the *local density* , $d_j(S')$ for column j, $1 \leq j \leq n$, be the number of sets where $j \in Q_i$. Thus, the local density is the number of distinct nets in column j. The *density* is the maximum value of $d_j(S')$. Let the $D(S')$ represent the density for intervals in the set S'. $D(S)$ is called the *channel density*. The channel density is also defined by the number of vertices in the maximal clique in HG. Since $D(S)$ represents the number of nets that satisfy condition (ii), the channel density is the minimum value of k necessary to complete the routing.

3.2. Channel Assignment in Optimal Solutions

Let the *track* Tr_i, for $1 \leq i \leq k$, be the set of intervals mapped to i by δ, the channel assignment mapping. Let S_i denote the set of intervals



$N_1=\{(0,left),(1,top),(4,top)\}$
$N_2=\{(1,bot),(4,bot),(5,top)\}$
$N_3=\{(3,top),(5,bot)\}$
$N_4=\{(2,top),(2,bot)\}$

$P_1=\{[0,1]^1,[1,4]^1\}$
$P_2=\{[1,4]^2,[4,5]^2\}$
$P_3=\{[3,5]^3\}$
$P_4=\{\}$

$Q_1=\{0,1,2,3,4\}$
$Q_2=\{1,2,3,4,5\}$
$Q_3=\{3,4,5\}$
$Q_4=\{\}$

$d_1(S)=2$
$d_2(S)=2$
$d_3(S)=3$
$d_4(S)=3$
$d_5(S)=2$
$D(S)=3$

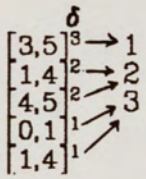


Figure 3.1. Formalization of a simple channel

remaining to be mapped after i tracks have been completed. Hence, $S_0=S$.

The following Theorem characterizes the number of tracks required in a channel routing solution. The theorem assumes the mapping of intervals in track $i-1$ is completed before any intervals are mapped to track i.

Theorem 3.1: The number of tracks, k, in a channel routing solution is equal to the channel density, $D(S_0)$, plus the number of tracks where $D(S_{i-1})=D(S_i)$.

Proof

The densities, $D(S_i)$ after the selection of intervals in track Tr_i form the non-increasing sequence

$$D(S_0)\geq D(S_1)\geq D(S_2), \ldots, \geq D(S_k)=0$$

Because of this and condition(ii)'s guarantee that the intervals in distinct nets do not overlap,

$$0\leq D(S_{i-1})-D(S_i)\leq 1.$$

Thus, the selection of intervals for Tr_i must lower the density of the remaining intervals by either 1 or 0. Clearly the density must be lowered exactly $D(S_0)$ times and repeated $k-D(S_0)$ times.

Consider the example of Figure 3.2. Track Tr_1 is at the bottom of the channel and Tr_4 is at the top. The density ordering is $3\geq 2\geq 2\geq 1\geq 0$. $D(S_0)=3$, the number of tracks where $D(S_{i-1})=D(S_i)$ is one, and $k=4$.

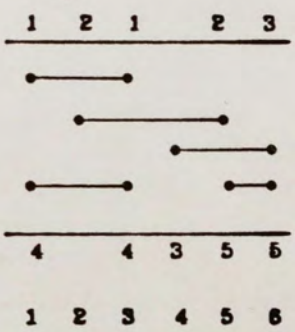


Figure 3.2. Theorem 3.1. example

A selection of intervals for track Tr_i is called *density reducing* if $D(S_{i-1})-D(S_i)=1$. If each track Tr_i is density reducing, the number of times $D(S_{i-1})=D(S_i)$ must be zero and solution must have exactly $D(S_0)$ tracks. This result implies the following Lemma:

Lemma 3.1: A solution to a given channel routing problem has $k=D(S_0)$ tracks if and only if for each track Tr_i where $1 \le i \le k$, $D(S_{i-1})-D(S_i)=1$.

Theorem 3.1 and Lemma 3.1 are used frequently in this chapter. In section 3.3, Lemma 3.1 is used to show that the Left edge algorithm always produces an optimal solution in the absence of vertical constraints. In section 3.6, Theorem 3.1 is used to analyze conditions that lead to the Dogleg algorithm's failure to complete the channel in the channel density. This analysis leads to two new channel routing algorithms with significantly higher completion rates than the Dogleg algorithm.

3.3. The Left Edge Algorithm

This section presents a formal description of the *Left Edge* algorithm and proves that it produces an optimal solution in the absence of vertical constraints.

Let L be an ordering of S where $L_j=[a_j,z_j]^t$ and $L_{j+1}=[a_{j+1},z_{j+1}]^t$ denote the j^{th} and $j+1^{th}$ elements of L such that $a_j \le a_{j+1}$ for $1 \le j \le (|S|-1)$. Thus, the set of intervals L is ordered by ascending left endpoint.

The Left Edge algorithm is shown below.

The Left Edge Algorithm

```
1   i = 0
2   while (L≠φ){
3     i = i + 1
4     for each 1≤j≤|L| {
5       HCG = true iff Tr_i=φ or (p,L_j)∉E_hg for p∈Tr_i
6       if HCG
7         Tr_i = Tr_i ∪ L_j
8         }
9     L=L−Tr_i
10    }
```

For each track Tr_i, the left edge algorithm scans the ordered set L for intervals that are compatible (in terms of HG) to previous segments placed in that track. Each compatible interval is added to Tr_i. Intervals from Tr_i are then removed from S.

Consider the sample channel in Figure 3.3(a). L initially contains $([0,3]^3,[1,7]^1,[2,4]^2,[5,8]^4,[6,9]^5)$. After the first pass of L, Tr_1 contains $\{[0,3]^3,[5,8]^4\}$. The set L after the deletion of these vertices is $([1,7]^1,[2,4]^2,[6,9]^5)$. The process continues until all intervals have been placed to yield $Tr_2=\{[1,7]^1\}$ and $Tr_3=\{[2,4]^2,[6,9]^5\}$. Figure 3.3(b) shows the completed channel.

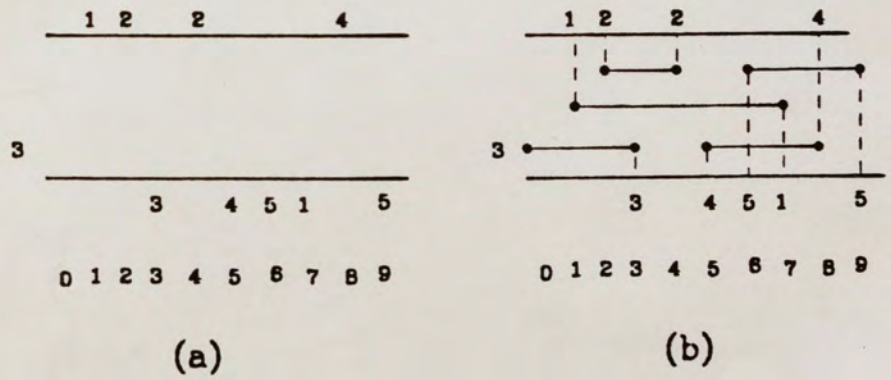


Figure 3.3. Left edge example

The complexity of the Left Edge algorithm is a function of b, the number of intervals in L. During each pass i of the left edge algorithm, line 5 guarantees that the first interval, L_1, is placed in Tr_i. The worst case performance occurs when no other intervals can be placed in track i. Thus, the number of times the inner loop is executed is $b+(b-1)+(b-2)...+1$ or $\frac{b(b-1)}{2}$. The inner loop involves determining if the current interval, L_j is compatible with other elements in Tr_i. The complexity of this step is considerably simplified by noting that if the current interval is compatible with the last interval added to Tr_i, the interval is also compatible with all other intervals in the track. Since the creation of HG and L also require no more than $O(b^2)$, the worst case Left Edge complexity is $O(b^2)$.

Corollary 3.1: Given a channel routing problem with no vertical constraints, the Left Edge algorithm always produces an optimal solution with $k=D(S_0)$.

Proof

Lemma 3.1 guarantees that if each selection of Tr_i is density reducing, the solution is optimal. By contradiction, let Tr_i be the first track where $D(S_{i-1})-D(S_i)=0$. Let column j be the first column such that $D(S_{i-1})=d_j(S_i)$, the local density in column j. This is the first column at the density that is not included in the set of intervals Tr_i selected by the left edge algorithm. Since all columns to the left of column j have a density less than $d_j(S_i)$, there must be some interval that begins in column j. But since the density did not decrease in this column, the interval must have been compatible with all previous intervals selected for this track, and thus would

have been selected by the left edge algorithm, a contradiction.

The proof that the Left Edge algorithm is optimal in the absence of vertical constraints was first shown by Hashimoto and Stevens[HaSt71]. LaPaugh[La80] showed that the interval assignment problem under these conditions is equivalent to the interval coloring problem, which was shown to be solvable in polynomial time[Ga72].

3.4. The Basic Dogleg Algorithm

This section formally describes the Dogleg algorithm and, using Lemma 3.1, determines under what conditions the Dogleg algorithm fails to complete the channel in a number of tracks equal to the density.

The Basic Dogleg algorithm is much the same as the Left Edge algorithm presented earlier. The algorithm presented here fills tracks from left to right, starting with track 1, then track 2, etc. until all intervals have been placed. As in the Left Edge algorithm, L denotes the set of intervals in S ordered by ascending left endpoint and L_j denotes the j^{th} element in L.

The Basic Dogleg algorithm is shown below.

The Basic Dogleg Algorithm

```
1   i = 0
2   while (L≠φ) {
3     i = i + 1
4     for each 1≤j≤|L| {
5       HCG = true iff Tr_i=φ or (p,L_j)∉E_hg for p∈Tr_i
6       VCG = true iff (L_j,q)∉E_vg for q∈V_vg
7       if HCG and VCG {
8         Tr_i = Tr_i ∪ L_j
9         VG = VG - L_j
10        }
11      }
12    L=L−Tr_i
13    }
```

This algorithm is very similar to the left edge algorithm with the exception of lines 6, 7 and 9. Line 6 assigns the boolean variable VCG a true value if and only if L_j has no descendants in the vertical constraint graph. If an interval has descendants in the vertical constraint graph, condition (i) of the channel routing problem requires that all descendents be assigned before this interval is selected. Line 9 removes a selected interval from the vertical constraint graph. This removal process implies the removal of any edges into or out of that vertex. As an illustration of this algorithm, consider the channel and vertical constraint graph of Figure 3.4(a). L initially contains ($[1,3]^1$,$[2,4]^3$,$[2,4]^5$,$[4,6]^5$, $[5,8]^4$,$[5,7]^2$). After the first pass, Tr_1 contains {$[1,3]^1$,$[5,8]^4$}; L now contains ($[2,4]^3$,$[2,4]^5$,$[4,6]^5$,$[5,7]^2$), and the vertices corresponding to nets 1 and 4 have been deleted from the vertical constraint graph. In the second pass, intervals {$[2,4]^3$,$[5,7]^2$} have no ancestors in the vertical constraint graph. Both are selected in the left to right pass of L and placed in Tr_2. Figure 3.4(b) shows the track selections made by the Basic Dogleg algorithm for the remainder of the channel.

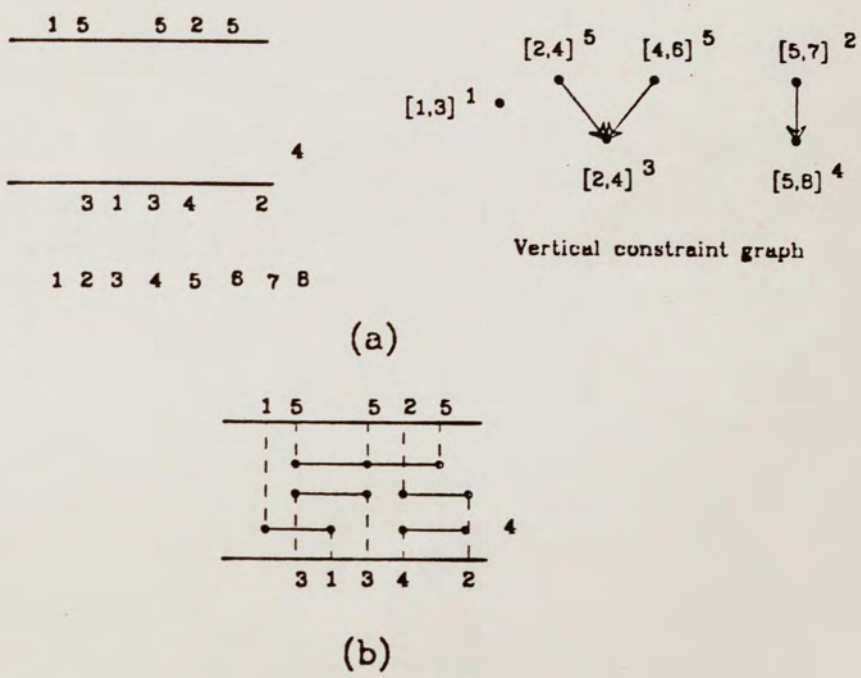


Figure 3.4. Basic Dogleg example

Since the vertical constraint graph is acyclic, there must be at least one interval with no descendants that can be selected for each track. Since there are a finite number of intervals, b, the Basic Dogleg algorithm always finds a solution with at most b tracks. The worst case performance of the Basic Dogleg occurs when only one interval is selected for placement on each track. As in the Left Edge, the worst case performance is $O(b^2)$.

3.5. The Dogleg Algorithm

Deutsch's statement of the Dogleg algorithm has two differences compared to the Basic Dogleg algorithm presented above:

(1) Deutsch's algorithm alternates between the filling of top and bottom tracks

(2) Deutsch's algorithm fills top tracks from right to left and bottom tracks from left to right

The *Dogleg* algorithm, presented below, incorporates the extensions specified by Deutsch.

As in the Left Edge algorithm, L denotes the set of intervals in S ordered by ascending left endpoint, and L_j denotes the j^{th} element in L. Similarly, R denotes the set of intervals in S ordered by descending right endpoint.

The extended Dogleg algorithm is shown below.

The Dogleg Algorithm

```
 1   i = 0
 2   while (L≠φ) {
 3     i = i + 1
 4     for each 1≤j≤|L| {
 5       if i is odd {
 6         s=L_j
 7         VCG = true iff Tr_i=φ or (L_j,q)∉E_vg for q∈V_vg
 8         }
 9       else {
10         s=R_j
11         VCG = true iff Tr_i=φ or (q,R_j)∉E_vg for q∈V_vg
12         }
13       HCG = true iff (p,L_j)∉E_hg for p∈Tr_i
14       if HCG and VCG {
15         Tr_i = Tr_i ∪ s
16         V_vg = V_vg - s
17         }
18       }
19     L=L-Tr_i
19     R=R-Tr_i
20     }
```

To extend the basic dogleg algorithm to alternate between the filling of top and bottom tracks, lines 6-7 are executed when the track index, i, is odd, indicating a bottom track; and lines 11-12 are executed when the track index is even, filling a top track. In the case of a bottom track, line 7 assigns a boolean value to VCG indicating that the current

interval, s, has no descendants in the vertical constraint graph. Similarly, line 11 indicates that s has no ancestors in the vertical constraint graph.

To extend the algorithm to make a left to right pass on bottom tracks and a right to left pass on top tracks, intervals are chosen from L on bottom tracks and from R on top tracks. Selecting intervals sorted by increasing left end-point is called a *left edge* pass while selecting intervals sorted by decreasing right endpoint is called a *right edge* pass.

Consider the sample channel routing problem given in Figure 3.5. L initially contains ($[1,3]^1$,$[2,4]^3$,$[2,4]^5$,$[4,6]^5$, $[5,8]^4$,$[5,7]^2$). R contains ($[5,8]^4$,$[5,7]^2$,$[4,6]^5$,$[2,4]^5$, $[2,4]^3$,$[1,3]^1$). For the first pass, i is odd; the

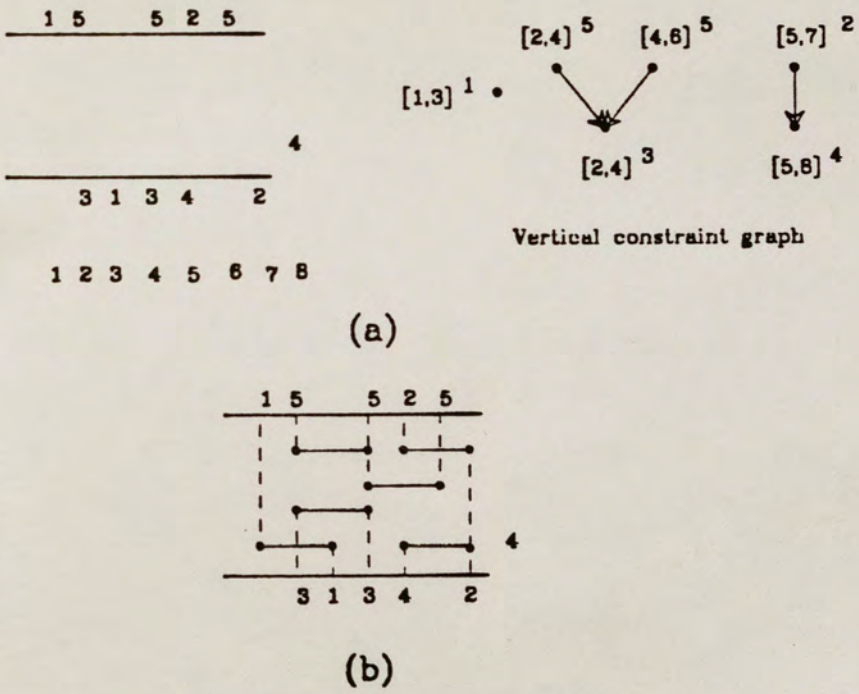


Figure 3.5. Dogleg algorithm example

pass is left to right; and the intervals are selected from set L. Since odd values of i imply that the track filled is a bottom track, line 7 determines if the interval s has any descendants in the vertical constraint graph. Thus, at the end of the first pass, Tr_1 contains $\{[1,3]^1,[5,8]^4\}$. These intervals are deleted from the vertical constraint graph and the process continues, filling Tr_2 with intervals destined for the top track using the ordered set R. The intervals selected for the top track are $[5,7]^2$ and $[2,4]^5$. The final channel produced by the Dogleg algorithm is shown in Figure 3.5(b). Notice that the track index i no longer corresponds directly to the track number although it is not difficult, once the algorithm has completed, to rearrange the track indices to correspond to physical track numbers.

Since each pass through the algorithm uses either a left edge algorithm or a right edge algorithm, the Dogleg algorithm's worst case complexity is identical to the Left edge algorithm; the worst case complexity is $O(b^2)$.

The Dogleg algorithm uses a symmetric approach to choosing tracks and intervals to be filled as opposed to the bottom to top, left to right approach of the Basic Dogleg algorithm. Does this approach decrease the average number of tracks in channel routing solutions? For example, in Figure 3.4 the Basic Dogleg completed the sample channel in only 3 tracks whereas the Dogleg algorithm produced a solution using 4 tracks. Although a formal description of the generation of test cases and results is postponed until Chapter 5, the evidence indicates that the Basic Dogleg and the Dogleg algorithms require approximately the same number of tracks to complete randomly generated sample channels. If these statistics are representative, why use the more complicated Dogleg algorithm? One

reason is that the Dogleg algorithm, by alternating between top and bottom tracks, tends to minimize the length of the vertical wiring. For example in section 5.4, the Basic Dogleg used 13% more vertical wire than the Dogleg algorithm did. This increases the capacitance of the integrated circuit, decreases speed and increases power consumption.

3.6. An Analysis of Dogleg Generated Solutions

Theorem 3.1 guarantees one extra track must be added to the channel for each selection of intervals in track Tr_i that is not density reducing, i.e. where $D(S_{i-1})-D(S_i)=0$. This section uses this result in determining when the Basic Dogleg and Dogleg algorithms fail to complete the wiring in the channel density.

Figure 3.6 steps through the Basic Dogleg algorithm using an example that does not complete in the density. Figure 3.6(a) shows a sample channel and the corresponding vertical constraint graph. The density of the channel is 3, yet the Basic Dogleg and Dogleg algorithms complete the channel in 4 tracks, as shown in Figure 3.6(b). Both algorithms choose intervals $[1,3]^1$ and $[6,7]^4$ for track 1. Notice that these intervals are not density reducing. Theorem 3.1 guarantees that this selection of intervals will increase by one the number of tracks necessary to complete the channel.

Had the Basic Dogleg algorithm used a right edge rather than a left edge algorithm, the channel would have completed using only the 3 tracks shown in Figure 3.6(c). Clearly the order (top or bottom) in which tracks are chosen to be filled and the direction (left edge or right edge) each track is filled affects the number of tracks in the final solution. In Chapter 5 results are presented that indicate that the

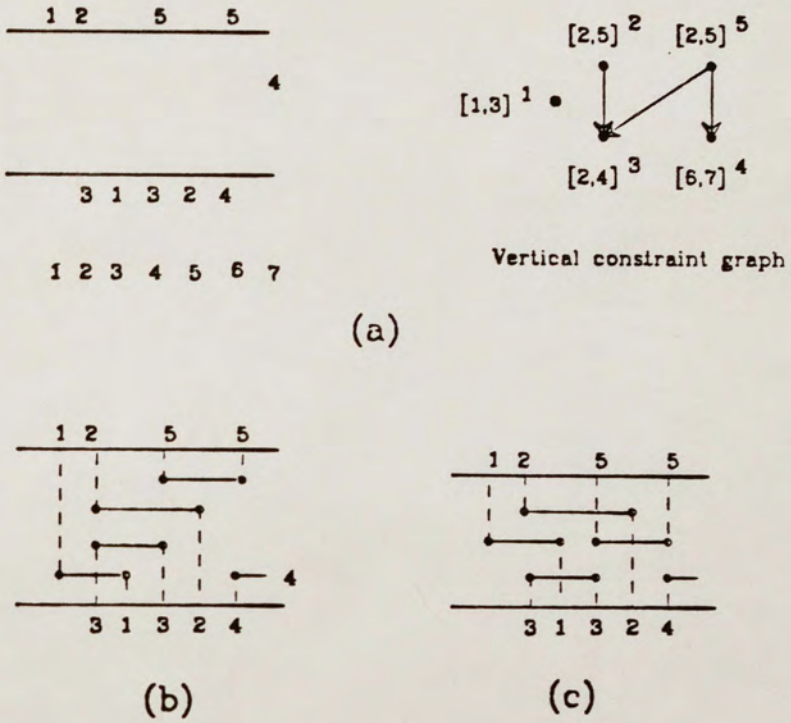


Figure 3.6. Routing failure example.

Basic Dogleg and Dogleg algorithms use almost an identical number of tracks over a large number of randomly generated channels. Since each algorithm uses different track selection and direction schemes, any particular choice of ordering merely changes the solution; it does not necessarily improve it. The Bell Labs LTX[PeDeSc76] system tries several possible permutations, saving the choice that results in the lowest number of tracks. Since the number of such permutations grows exponentially, using random permutations requires repeated application of the Dogleg algorithm and does not guarantee that a favorable ordering will be among the permutations tried.

3.7. A Revised Dogleg Algorithm

The Dogleg algorithm fixes the order tracks are selected and the direction each track is filled. This section uses Theorem 3.1 as the basis for heuristics that substantially improve the Dogleg algorithm's performance.

Theorem 3.1 guarantees that if the selection of intervals to be placed on a track is not density reducing, the selection will increase by one the number of tracks in the completed channel. The *Revised* Dogleg algorithm attempts to maximize density reducing track selections by analyzing four possible track selections.

The alternatives attempted are
(1) left edge on a bottom track
(2) right edge on a bottom track
(3) right edge on a top track
(4) left edge on a top track

Consider the example given in Figure 3.7. Figure 3.7(a) shows the intervals that need to be placed and the values for the local densities, $d_i(S_0)$. Figure 3.7(b-d) shows each of the track selection alternatives along with the local densities after those intervals have been eliminated. Figure 3.7(b) corresponds to a left edge pass on a bottom track; Figure 3.7(c), a right edge pass on a bottom track; Both the left edge and the right edge passes on the top track produce the track selection shown in Figure 3.7(d). Clearly choices c and d are better than b since each is density reducing.

The Revised algorithm makes both a left edge and right edge pass of the intervals that can be placed on the top track, followed by a left edge and a right edge pass of intervals that can be placed on the bottom track. The notation Tr_{ll} is used to represent the track containing intervals chosen for a top track using a left edge pass of

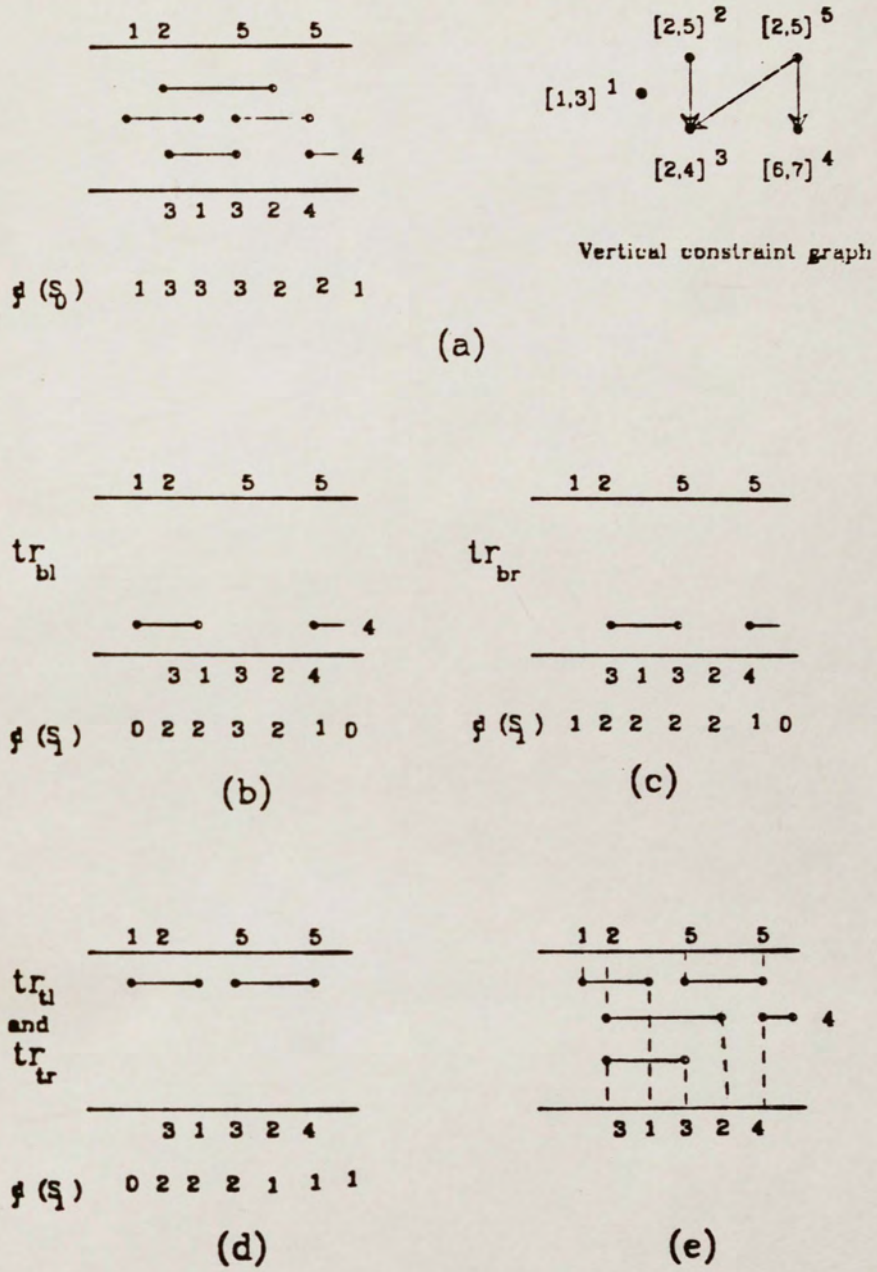


Figure 3.7. Revised algorithm track selection example

the available intervals. Similarly, track Tr_{br} represents a track that contains intervals chosen for a bottom track using a right edge pass, etc.

The Revised Dogleg algorithm is shown below.

The Revised Dogleg Algorithm

```
1   i = 0
2   while (L ≠ φ) {
3     i = i + 1
4     for each 1≤j≤|L| {
5       s = L_j
6       VCG_top = true iff (L_j, q) ∉ E_vg for q ∈ V_vg
7       VCG_bot = true iff (q, L_j) ∉ E_vg for q ∈ V_vg
8       HCG = true iff Tr_i = φ or (p, s) ∉ E_hg for p ∈ Tr_i
9       if HCG {
10        if VCG_top
11        Tr_tl = Tr_tl ∪ s
12        if VCG_bot
13          Tr_bl = Tr_bl ∪ s
14        }
15      s = R_j
16      VCG_top = true iff (R_j, q) ∉ E_vg for q ∈ V_vg
17      VCG_bot = true iff (q, R_j) ∉ E_vg for q ∈ V_vg
18      HCG = true iff Tr_i = φ or (p, s) ∉ E_hg for p ∈ Tr_i
19      if HCG {
20        if VCG_top
21          Tr_tr = Tr_tl ∪ s
22        if VCG_bot
23          Tr_br = Tr_bl ∪ s
24        }
25      }
26    Tr_i = SELECT( L, Tr_tl, Tr_bl, Tr_tr, Tr_br)
27    R = R − Tr_i
28    L = L − Tr_i
29    V_vg = V_vg − Tr_i
30    }
```

Lines 5-14 fill a top track using both a left edge (Tr_{tl}) and a right edge (Tr_{bl}) pass. Similarly, lines 15-25 fill tracks Tr_{tr} and Tr_{br}. The SELECT function in line 26 attempts to choose a track that is density reducing.

The SELECT function is shown below.

SELECT function

SELECT ($L, Tr_{tl}, Tr_{bl}, Tr_{tr}, Tr_{br}$)

$\alpha_{tl} = |\{l \mid d_l(L) = D(L) \text{ for } 0 \le l \le n+1\} - Tr_{tl}|$

$\alpha_{tr} = |\{l \mid d_l(L) = D(L) \text{ for } 0 \le l \le n+1\} - Tr_{tr}|$

$\alpha_{br} = |\{l \mid d_l(L) = D(L) \text{ for } 0 \le l \le n+1\} - Tr_{br}|$

$\alpha_{bl} = |\{l \mid d_l(L) = D(L) \text{ for } 0 \le l \le n+1\} - Tr_{bl}|$

$\gamma_{tl} = |\{l \mid d_l(L) - d_l(L - Tr_{tl} = 1) \text{ for } 0 \le l \le n+1|$

$\gamma_{bl} = |\{l \mid d_l(L) - d_l(L - Tr_{bl} = 1) \text{ for } 0 \le l \le n+1|$

$\gamma_{tr} = |\{l \mid d_l(L) - d_l(L - Tr_{tr} = 1) \text{ for } 0 \le l \le n+1|$

$\gamma_{br} = |\{l \mid d_l(L) - d_l(L - Tr_{br} = 1) \text{ for } 0 \le l \le n+1|$

return track with minimum value of α. If two or more tracks have the same α return the track with the maximum value of γ.

end SELECT

A column j is *covered* by track Tr_i if $d_j(S_{i-1}) - d_j(S_i) = 1$. A column j is said to be *critical* if $d_j(S_{i-1}) = D(S_{i-1})$. Thus, a track is density reducing if each critical column is covered. The value of α represents the number of critical columns that are not covered by the selection of the intervals in the track. Thus, α can be thought of as a penalty for selecting a particular track assignment. If no track is density reducing, the SELECT function will choose the track that minimizes α.

If two or more tracks minimize α, the SELECT function chooses the track that covers the most number of columns. This is equivalent to choosing the track with the smallest amount of "white space". Since the algorithm is guaranteed to find an optimal solution if a density reducing selection of intervals can be selected during each pass, this heuristic attempts to minimize the number of critical columns by maximizing local density reductions in areas with non-critical columns.

Returning to Figure 3.7, α_{tr} and $\alpha_{tl} = 0$; $\alpha_{bl}=1$; and $\alpha_{br} = 0$. $\gamma_{tl}=\gamma_{tr}=6$; $\gamma_{bl}=5$; and $\gamma_{br}=5$. Tracks Tr_{tr}, Tr_{tl} and, Tr_{br} all have equal values for α. Since Tr_{tl} and Tr_{br} maximize γ, this choice is used for Tr_i. This process continues until all the intervals have been placed. Figure 3.7(e) shows the completed channel.

Chapter 5 compares the completion rates of the Revised channel routing algorithm with the Dogleg algorithm over a variety of randomly generated channels. The results indicate that the Dogleg algorithm uses approximately four times the number of additional tracks as the Revised algorithm does.

The revised algorithm requires that, during each pass, four possible tracks be attempted. This requires each interval to be examined 4 times. This is a linear increase in the worst case performance of the Dogleg algorithm. Thus, the complexity of the Revised algorithm is also $O(b^2)$.

3.8. The Least Cost Path (LCP) Algorithm

The Revised Dogleg algorithm attempts to map a set of density reducing intervals to a track. To accomplish this the algorithm makes both a left edge and right edge pass of the intervals that can be placed on the top track, followed by a left edge and a right edge pass of intervals that can be placed on the bottom track. As the results in Chapter 5 will confirm, this approach significantly improves the number of tracks over the density required to complete the channel. Unfortunately, there exist channels where none of the revised algorithm's passes will uncover a combination of intervals that is density reducing, even if in the remaining intervals, such a set exist.

Consider the example in Figure 3.8(a). In making the choice of intervals for the first track, the left edge pass always selects $[1,3]^7$, and the right edge pass always selects $[8,10]^1$. Neither of these two choices is density reducing; thus, by Theorem 1, the routing will always exceed the density. The completed channel produced by the Revised Algorithm is shown in Figure 3.8(b). Figure 3.8(c) shows that

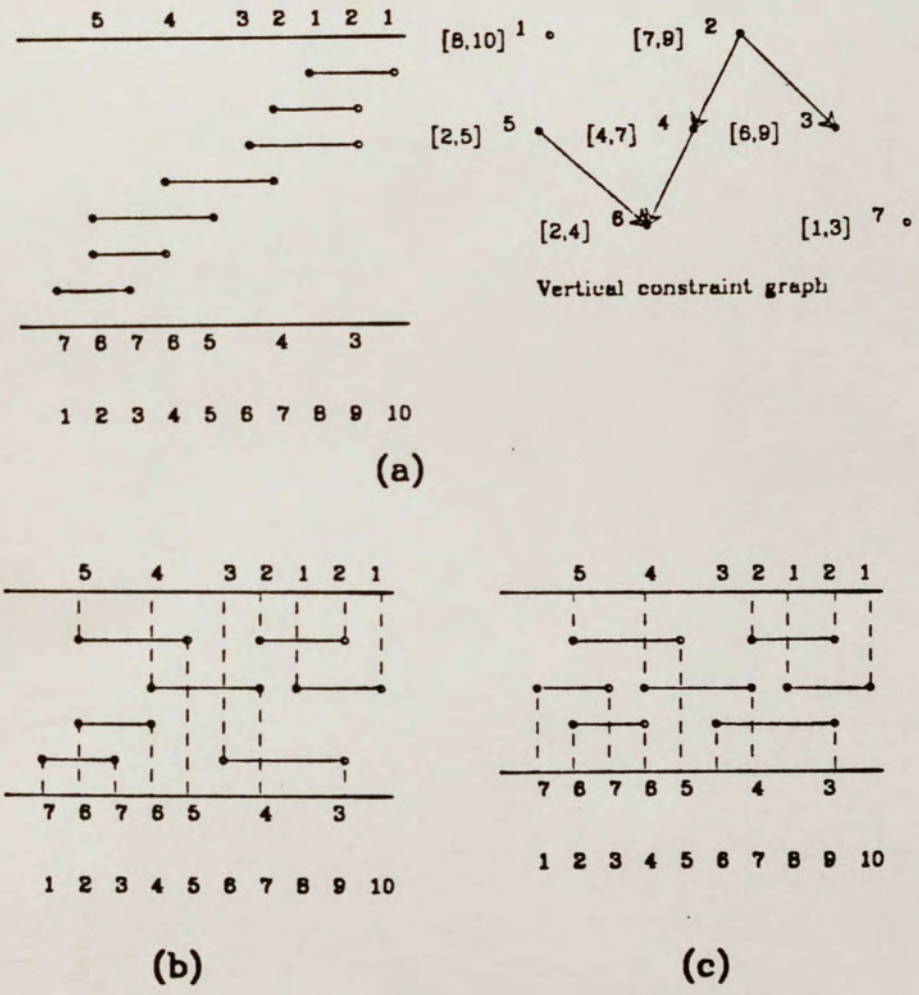


Figure 3.8. Revised algorithm misrouting

there is a set of intervals that is density reducing that was not considered by the Revised algorithm.

In this section we develop a new Least Cost Path (LCP) algorithm. This algorithm is shown in Chapter 5 to yield significantly improved results. The LCP algorithm guarantees that if a set of intervals contains a density reducing set of intervals, then those intervals will be selected for the current track.

The selection process involves finding the least cost path through two directed weighted acyclic graphs: one for the intervals that have no descendants in the vertical constraint graph and one for the intervals with no ancestors. The paths through each graph correspond to possible combinations of intervals eligible for the current track. Two pseudo-intervals are added to the graph that act as a source and sink for the path. The source is represented by the interval $v_{source}=[0,0]^0$, and the sink, by $v_{sink}=[n+1,n+1]^0$. The LCP algorithm attempts to select a path that is density reducing.

Let the directed graph $GT(V_{top},E_{top})$ be defined for some set of intervals $S'\subseteq S$ as follows:

$$V_{top}=\{v_{source},v_{sink}\}\cup\{v\,|\,(q,v)\notin E_{vg}\ for\ q,v\in V_{vg}\}$$

$$E_{top}=\{(p=[a,z]^i,q=[a',z']^j)\,|\,(p,q)\notin E_{hg}\ and\ a\le a'\ for\ p,q\in V_{top}\}$$

Similarly, $GB(V_{bot},E_{bot})$ is defined as

$$V_{bot}=\{v_{source},v_{sink}\}\cup\{v\,|\,(v,q)\notin E_{vg}\ for\ q,v\in V_{vg}\}$$

$$E_{bot}=\{(p=[a,z]^i,q=[a',z']^j)\,|\,(p,q)\notin E_{hg}\ and\ a\le a'\ for\ p,q\in V_{bot}\}$$

Associated with each edge, $e=(p=[a,z]^i,q=[a',z']^j)$, in E_{top} and E_{bot}, let the cost-function $C(e)=\left|\left\{l\,|\,d_i(S_{i-1})=D(S_{i-1})\ for\ z<l<a'\right\}\right|$. Thus, $C(e)$ maps each edge in E_{top} to the number of columns between intervals p

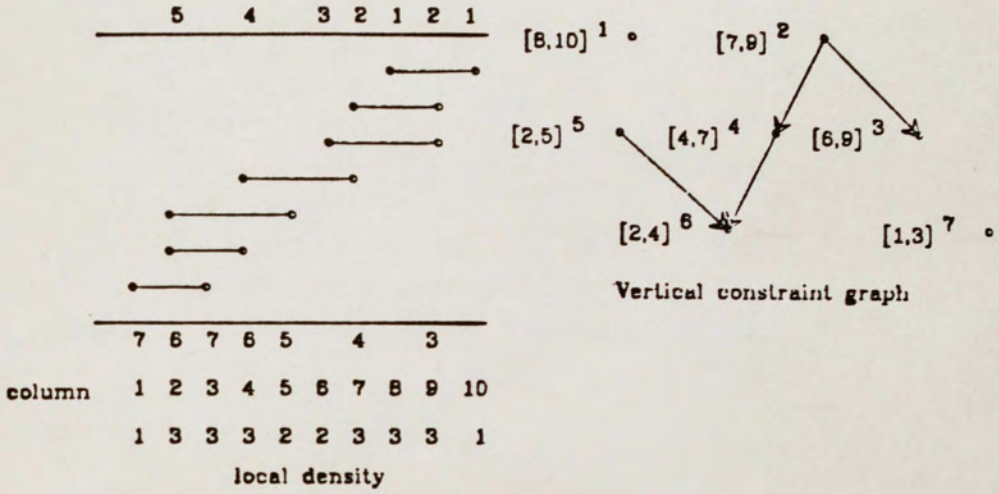
and q that are critical. Let $Y=v_{source},v_1,v_2,\ldots,v_{sink}$ be a path in GT or GB. Let $W(Y)$ be the sum of the weights of the edges in path Y. Thus, for the intervals in Y to be density reducing, $W(Y)$ must be equal to zero. $W(Y)$ is equivalent to α, the number of critical columns not covered by the Revised algorithm in the SELECT function.

As an example, Figure 3.9 repeats the channel in Figure 3.8 and shows the graphs for GT and GB.

The least cost path, Y, through GT is $[0,0]^0,[2,5]^5,[7,9]^2,[11,11]^0$ and $W(Y)=0$.

The number of vertices in GT and GB is a subset of the number of intervals, b. The number of edges in a directed acyclic graph is $O(b^2)$. Thus, the creation of GT and GB is $O(b^2)$. Finding the shortest path through a graph using Dijkstra's algorithm is $O(|E|)$, the number of edges. Thus, the worst case time complexity necessary to create the graph and find the shortest path is also $O(b^2)$. In practice, the least cost path can be determined during the construction of GT and GB.

The LCP algorithm can now be stated in terms of these two graphs. Let P_{top} and P_{bot} be the minimum weighted paths in GT and GB respectively. Two or more paths are *density equivalent* if they have the same weighted cost, i.e. they cover the same number of columns that are at the density. If two or more paths are density equivalent, the path is chosen that covers the most columns. This additional heuristic is identical to that used in the Revised algorithm described earlier.



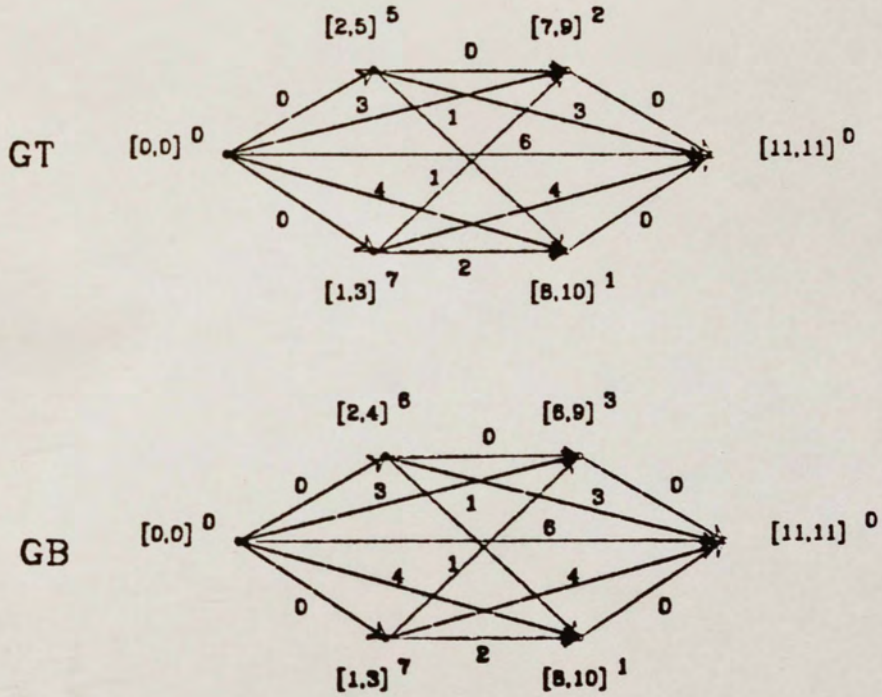


Figure 3.9. Weighted Graph examples

The complete LCP algorithm is shown below.

The Least Cost Path Algorithm

```
1   i = 0
2   while (S ≠ φ) {
3     i = i + 1
4     Construct GT and find P_top
5     Construct GB and find P_bot
6     Let P be a path of P_top and P_bot
7       with minimum cost W(P), and
8     maximum column coverage.
9     Tr_i = P
10    S = S − Tr_i
11    V_vg = V_vg − Tr_i
12    }
```

In Figure 3.9, P_{top} is $[0,0]^0,[2,5]^5,[7,9]^2,[11,11]^0$ and P_{bot} is $[0,0]^0,[2,4]^6,[6,9]^3,[11,11]^0$; both of which are density equivalent. The intervals in each path cover 7 columns. Thus, the choice of which path to choose for the first track is arbitrary; otherwise, the path that maximizes the number of columns covered would be selected. This process continues until all the intervals have been processed. In the next pass of the LCP algorithm, the *GT* and *GB* graphs are recomputed and the least cost path is again selected. This process continues until all intervals have been placed. The completed channel is shown in Figure 3.10.

If a combination of intervals exists that reduces the density of the remaining problem, the LCP algorithm will find it since *GT* and *GB* contain all possible combinations of intervals that can be placed on the current track. The channel routing problem is still, of course, NP complete. If neither P_{top} nor P_{bot} is density reducing, there is no guarantee that a different choice of density equivalent paths in earlier tracks would not have resulted in a density reducing choice of

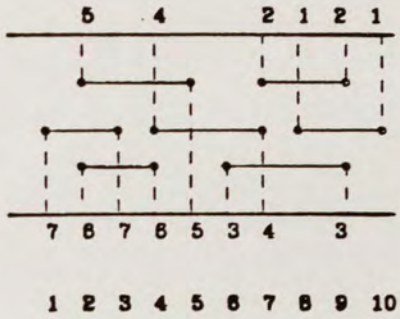


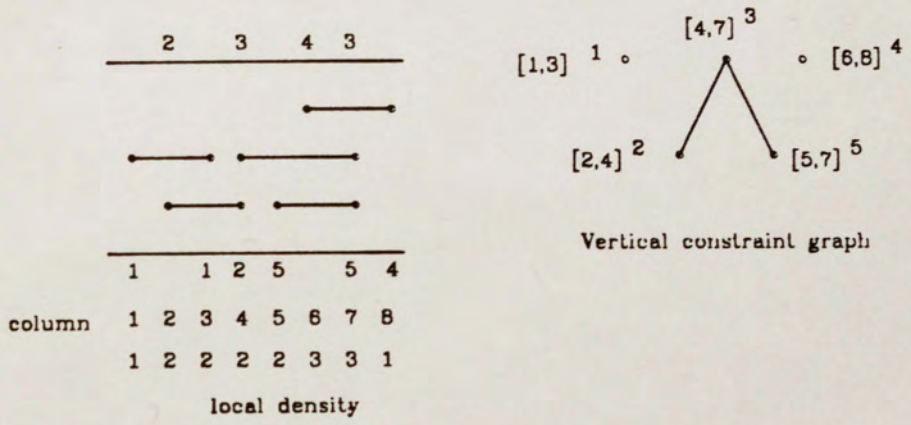
Figure 3.10. Channel selection for the LCP algorithm

intervals in the current track. To guarantee an optimal solution, backtracking must be used to reconsider previous arbitrary choices.

The complexity of the LCP algorithm is proportional to the number of times through the loop times the time necessary to construct and find the shortest path through *GT* and *GB*. Thus, the LCP complexity is $O(b*b^2)$ or $O(b^3)$. Although the worst case complexity is greater than the previous algorithms, in practice, the algorithm performs very well.

The LCP algorithm presented above uses the number of columns covered in the track to select between density equivalent paths. This choice can be improved by considering a measure of the importance of covering a particular column. Let j be a column with a density of $d_j(S')$. If $d_j(S')=1$, only one interval must be mapped into $D(S')$ tracks; if $d_j(S')=D(S')-1$, then $D(S')-1$ intervals must be mapped into $D(S')$ tracks. Clearly it is more important to place intervals that reduce those columns with the highest local density. Lines 4,5 and 6-8 of the LCP algorithm involve choosing between density equivalent paths. To incorporate the importance of the column, the path with the lowest ***weighted*** covering is selected. The weighted covering is the sum of

$d_j(S')$ for all columns j not included in the intervals in a path P. Thus, if column j is not selected, the "penalty" associated with that choice is equal to the local density in that column. Consider Figure 3.11.



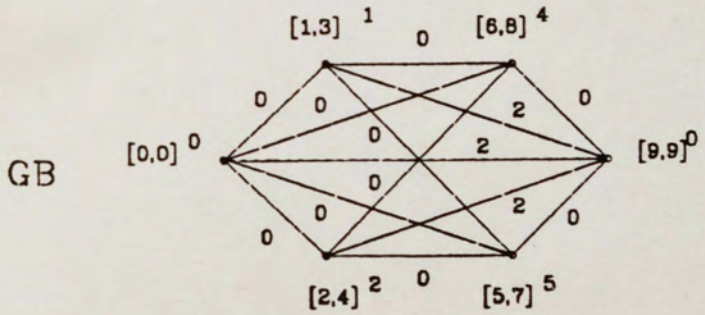


Figure 3.11. Weighted covering example

The possible density reducing paths through *GB* are shown below.

PATH	weighted cover
$P_1=[0,0]^0,[1,3]^1,[6,8]^4,[10,10]^0$	12
$P_2=[0,0]^0,[1,3]^1,[5,7]^5,[10,10]^0$	13
$P_3=[0,0]^0,[2,4]^1,[6,8]^4,[10,10]^0$	13
$P_4=[0,0]^0,[2,4]^1,[5,7]^5,[10,10]^0$	15
$P_5=[0,0]^0,[5,7]^5,[10,10]^0.$	8
$P_6=[0,0]^0,[6,8]^4,[10,10]^0.$	7

Given that $S=\{[1,3]^1,[2,4]^2,[4,7]_3,[6,8]^4,[5,7]^5\}$, the weighted covering for P_1 is then $d_1(S)+d_2(S)+d_3(S)+d_6(S)+d_7(S)+d_8=12$. Since P_4 maximizes the weighted covering, this path is selected.

The use of the LCP algorithm and the weighted cover heuristic form an extremely effective channel routing algorithm. As the results in Chapter 5 indicate, this algorithm used up to 20 times fewer tracks over the density than the standard Dogleg algorithm.

3.9. Applying Channel Routers to the Custom Routing Problem

The custom routing problem interconnects nets with fixed pins on the edges of randomly sized blocks. To apply channel routing algorithms to the custom routing problem, we must:

(1) Break up the routing region into channels.

(2) Select an ordering of the channels to be routed such that no floating pin's location becomes fixed before that channel is routed.

This section will show some of the difficulties with applying these steps to the general routing problem.

The first problem is to break up the routing region. Consider the example shown in Figure 3.12. Here the region has fixed pins on two

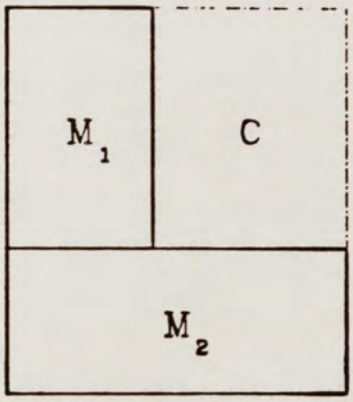


Figure 3.12. Difficult partitioning problem.

adjacent sides of the channel. Regions with this fixed pin organization are difficult if not impossible to break into channels.

Even if it is possible to partition the routing region into channels, there is no guarantee that it is possible to derive an appropriate order to route the channels. Consider the example in Figure 3.13(a). Here the channel C_1 must be routed before channel C_2, since the floating pins on the ends of C_1 cannot be fixed (as would happen if C_2 was routed first). A *preferred ordering* is an ordering of the channels such that ,when routed according to that order, the endpins of unrouted channels are not fixed. Thus, a possible preferred ordering is to route C_1 followed by C_2. A *channel graph* is used to determine a preferred ordering for a set of channels. Let $CG(V,E)$ be a directed graph where node $v_i \in V$ represents channel C_i and an edge $(v_i,v_j) \in E$ if channel C_i must be routed before channel C_j. Figure 3.13(b) shows the channel graph for a simple example. If the channel graph does not contain cycles, a preferred order can always be selected.

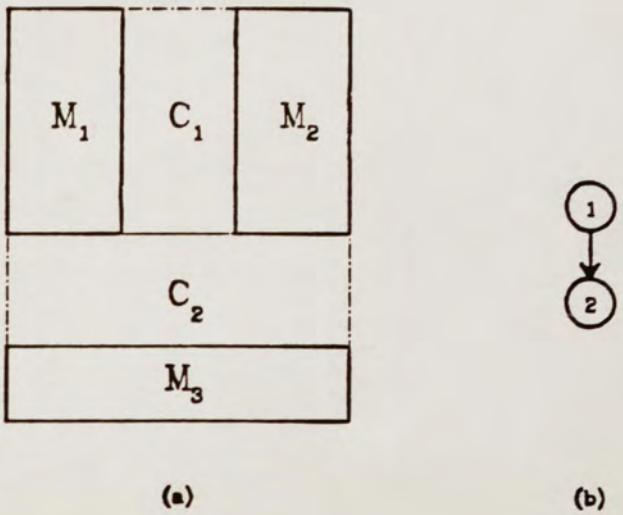


Figure 3.13. Channel ordering problem.

Unfortunately, it is not difficult to construct cases where no preferred ordering exists (i.e., there are cycles in the channel graph). Consider Figure 3.14(a). In this example, the only way to partition the problem into channels is as shown. Unfortunately, the 4 center channels form what is known as a *T loop*. The channel graph in Figure 3.14(b) has a cycle, indicating that the no preferred ordering exists.

Even if the region can be properly partitioned and a preferred channel ordering exists, good channel routing solutions are not guaranteed. Consider Figure 3.15(a). Here the routing region has been partitioned into channels C_1, C_2 and C_3. A preferred ordering also exists: C_1, C_3, C_2. Figure 3.15(b) shows the routing produced by the Dogleg in channels C_1 and C_3. For comparison purposes, Figure 3.15(d) illustrates the best possible routing for channel C_2.

The problem encountered in this example is that the crossings from channel to channel have not been judiciously chosen. Since

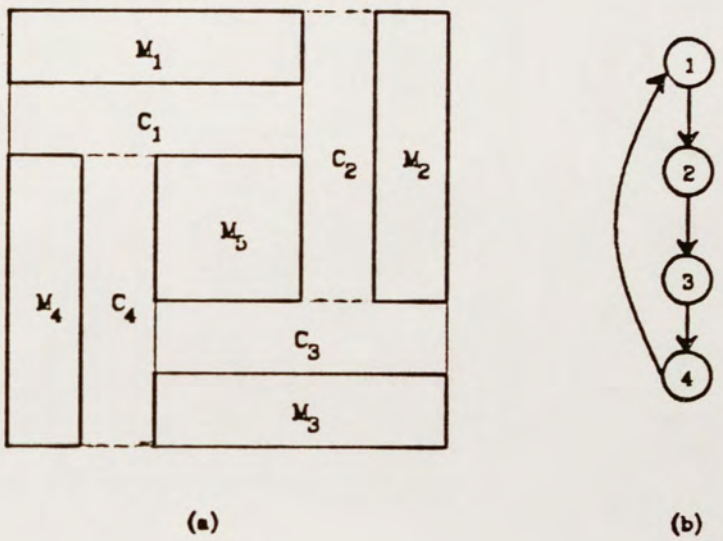


Figure 3.14. Problems with preferred ordering.

channel routers must be given the flexibility to choose the endpins, it is not possible to control the crossings between channels.

These examples indicate that channel routers alone are probably insufficient tools for solving the custom routing problem. Chapter 4 gives a detailed description area routing algorithms. Chapter 5 compares these algorithms with channel routing algorithms to determine which completes channels in the fewest number of tracks.

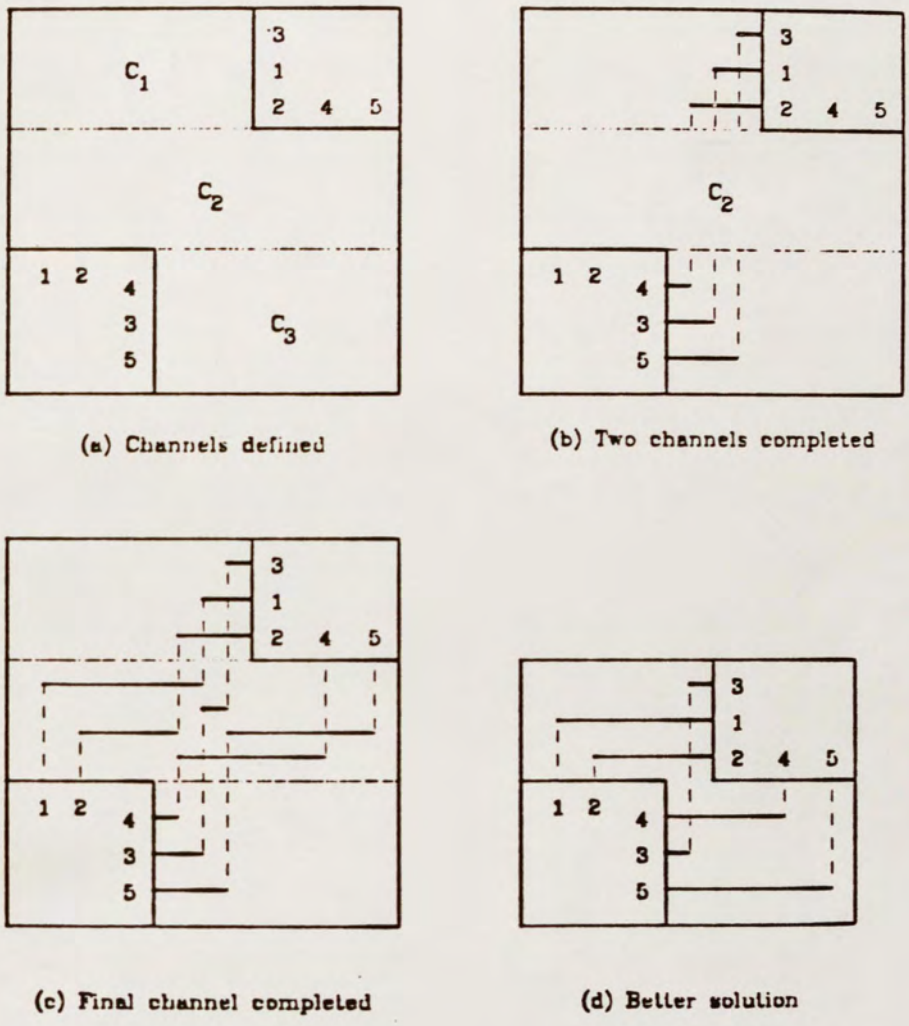


Figure 3.15. An example of poor channel routing performance.

The results will show that channel routers use fewer tracks than area routers when the region fits the definition of a channel. These results indicate that regions of custom integrated circuits that form natural channel should be routed as such, for the channel routing algorithms

use fewer tracks, are much faster area routing algorithms and guarantee 100% completion rates by allowing additional tracks to be added to the center of the channel.

3.10. Summary

This chapter formally described the channel routing problem. This formalization enabled us to succinctly describe the Left edge, Basic Dogleg and Dogleg algorithms. This formalization also led to the development of necessary and sufficient conditions for a channel to be completed in the channel density. Using Theorem 3.1 and Lemma 3.1, the Left edge algorithm was proved optimal and the Dogleg routing failures were analyzed. This analysis led to the development of the Revised and LCP algorithms. Chapter 5 compares the channel routing algorithms over a variety of randomly generated channels. The results will show clearly that the new algorithms significantly reduce the number of tracks over the density required to complete the channels. This chapter also discussed some of the pitfalls involved in using solely channel routers to complete the wiring of custom integrated circuits.

CHAPTER 4

Analysis of Area Routing Algorithms

A channel was defined in Chapter 3 as a region with fixed pins on the top and bottom of the region and "floating" pins on the right and left edges of the region. An *area* is a fixed sized rectangular region with fixed pins on any side of the region.

Consider an area with pins only on the top and bottom sides with the height of the region equal to k. Clearly if an area routing algorithm can determine if the channel can be completed in k tracks, the algorithm will provide a solution to the channel routing problem. Thus, the area routing problem must be at least as hard as the NP-complete channel routing problem.

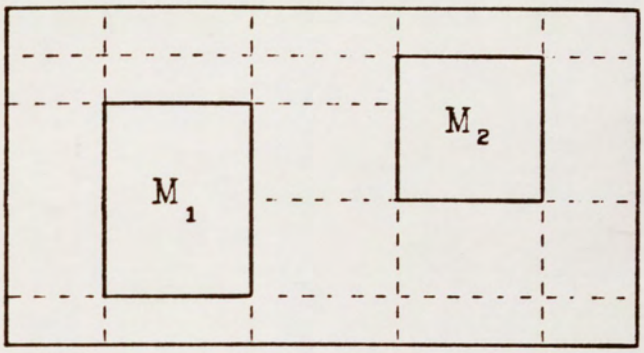
The previous chapter examined some of the problems associated with applying channel routing algorithms to the general routing problem. This chapter further motivates a discussion of area routing using the PI (Placement and Interconnect) project at MIT. This system depends upon area routers to complete the routing. The completion rate of routing algorithms is a function of the difficulty of the region. This chapter introduces a measure of the difficulty of a routing region. This measure is used to compare different area routing algorithms. The chapter also examines techniques used to minimize the effect of the net currently being completed on future unrouted nets. Finally, we introduce a new area routing algorithm that is functionally equivalent to the LUZ algorithm presented earlier, but has a much simpler implementation. The algorithm has several advantages over the LUZ algorithm. First, the new algorithm is extensible to

multipoint nets. Since multipoint nets are common in custom integrated circuits, this is an important extension. Second, the LUZ algorithm tried continually more complex paths, never knowing whether the path between source and destination is blocked. The new algorithm realizes when no path exists between source and destination and does not attempt to apply more complex path shapes; it simply halts and returns failure.

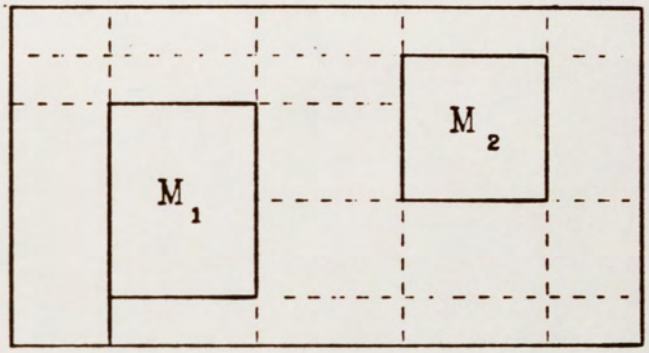
4.1. PI (Placement and Interconnect) Project

This section describes the partitioning of the routing region and the use of area routers in the PI (Placement and Interconnect)[Ri82a, Ba81] project. In this project the custom routing region is broken into "naturally" shaped areas, rather than channels. The algorithm performs this partitioning of the routing region by extending vertical and horizontal lines outward from the corner of each module, as shown in Figure 4.1(a). The heuristic used to combine the areas chooses the shortest line segment over all the lines extending from the corners. This line is selected as the boundary of one of the areas and all other lines extending from the endpoints of the chosen segment are removed. Figure 4.1(b) illustrates the selection and edge deletion step. Then, the next shortest line segment is selected, etc., until all areas have been defined. The final partition is shown in Figure 4.1(c).

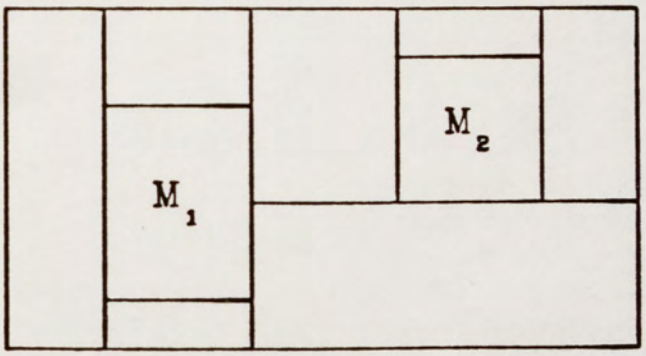
PI then uses a heuristic that fixes all crossings between areas. This approach allows global considerations to be included in the crossing placement, rather than allowing the crossings to be placed by the channel router. This has the side effect that every area has fixed pins on three or four sides. The result is a decomposition of the



(a) Corner extensions



(b) After the first edge is selected



(c) Completed channels

Figure 4.1. P.I. partitioning algorithm.

routing region into separate independent areas. PI then routes each area using one of several area routing algorithms.

The PI project has several limitations. First, it ignores the difficulties involved in placing the modules in the custom routing. How much space should be allowed between two modules? This question depends a great deal on how difficult the area is to route. If the routing is fairly simple, the placement of the blocks should leave only a small amount of area to complete the routing. In this section, a complexity measure called the *Manhattan Area Measure*, developed by Smith[Sm83], will be presented. The concept of measuring the difficulty of a routing region for use in the placement of modules required that the measure accurately model the wiring in the region. Smith's measure is inappropriate for this use because it deals strictly with two point nets. This chapter will extend the concept of area complexity measure to multipoint nets.

The problem of module placement is exacerbated by the inability of the PI system to dynamically update the region if necessary to complete the routing. Channel routers, however, can add tracks to the region to complete the routing.

As mentioned above, the PI system uses three different routers: a quick router (similar to LUZ 1 paths), a slice router (similar to the "greedy" channel router) and the Lee. Since the Lee is used as the router of last resort, we make the assumption that it completed the areas more often than the other two approaches. The quality of the PI solutions depends on the completion rates of the area routing algorithms. This chapter presents a new area routing algorithm, functionally equivalent to the LUZ, that is extensible to multi-point nets. The new algorithm is compared against the Lee over a variety of

randomly generated areas. The results, presented in Chapter 6, will show that the new algorithm has higher completion rates than the Lee and thus can improve the solutions produced by the PI system.

4.2. Manhattan Area Measure (MAM)

In area routing, the most important measure of an algorithm is its *completion rate*. The completion rate is defined as the number of times an algorithm completes all the routing divided by the number of problems attempted over a large number of samples. The completion rate is a function of the difficulty of the wiring problem. This function is qualitatively illustrated by the graph shown in Figure 4.2. The poor algorithm will have high completion rates only on very simple problems; a good algorithm will have high completion rates on much more difficult problems.

To measure the difficulty of the routing region, Smith[Sm83] proposed the use of the *Manhattan Area Measure* (MAM). The computation of the MAM for a given problem is as follows. For each

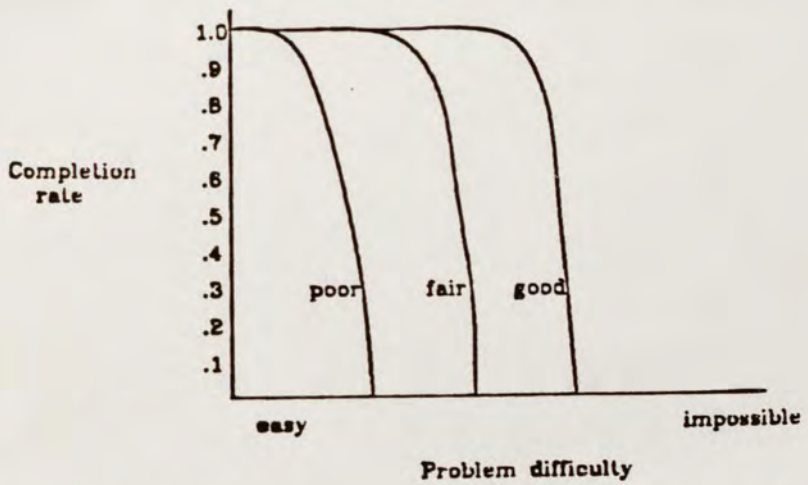


Figure 4.2. Graph of completion rates vs problem difficulty

net, compute the minimum number of grid points necessary to complete the net, independent of all other nets. For two point nets, this number is computed by calculating the manhattan distance between the closest grid point to each pin. This number corresponds to the number of edges in the path. Adding one to this number gives the number of grid points in the path. Then, the number of grid points for each net is summed and the result is divided by the number of grid points in the region.

The resultant number is the number of grid points that *must* be used to complete the routing. Multiple layers are accommodated by multiplying the number of grid points by the number of layers. Figure 4.3 gives an example. Here the MAM is computed as:

$$MAM = \frac{\sum grid\ points\ for\ each\ net}{\#\ grid\ points\ *\ \#\ layers}$$

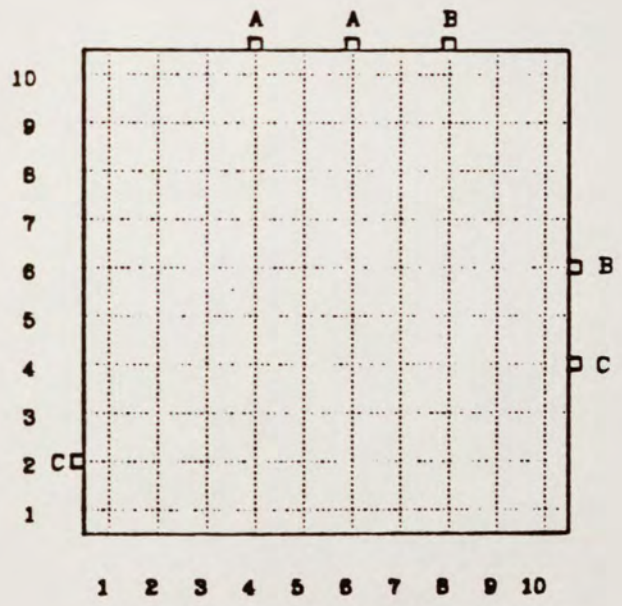


Figure 4.3. An example of the Manhattan Area Measure

Using Figure 4.3, the MAM is computed as follows:

$$\frac{3+(2+5)+(10+2)}{10*10*2} = .11$$

As the above example illustrates, a .11 MAM is a very sparse region. In contrast, the difficult example shown in Figure 2.6 that was completed by the LUZ had a MAM of .7.

Smith's definition of MAM is only applicable in two point nets. In later sections of this chapter the definition of MAM is extended to consider the more useful case of multi-point nets and revised to create tighter bounds using a two layer wiring model.

4.3. Area Routing Characteristics

If only one wire needs to be routed, the lower bound defined by the MAM is easily achieved. The difficulty in designing good routing algorithms is in optimizing the interaction between the different nets. An algorithm's approach to this interaction can be divided into two categories: *sequential* and *parallel* [Su82]. An algorithm is *sequential* if, when routing the i^{th} net, it only considers the placement of the $i-1$ nets already routed and some objective function for placing the net i. A *parallel* algorithm conceptually routes each wire independently of all other nets, and then resolves local conflicts by re-routing certain nets where there is over utilization of some grid edge. The Lee algorithm is an example of a sequential algorithm while most channel routing algorithms fall into the class of parallel algorithms.

A sequential algorithm must choose a path in such a way as to minimize the effect on nets yet to be routed. Four important

considerations are layer assignment, pin blocking, vias and degree of freedom.

4.3.1. Layer Assignment

Up to this point, all horizontal wires have been routed using one layer and all vertical wires using the other layer. This decision, when viewed locally, seems to increase the area required to complete the routing. For example, in Figure 4.4(a) nets 1 and 2 can be routed using only a single track with net one using the upper layer and net 2 using the lower layer. When viewed globally, this track sharing can

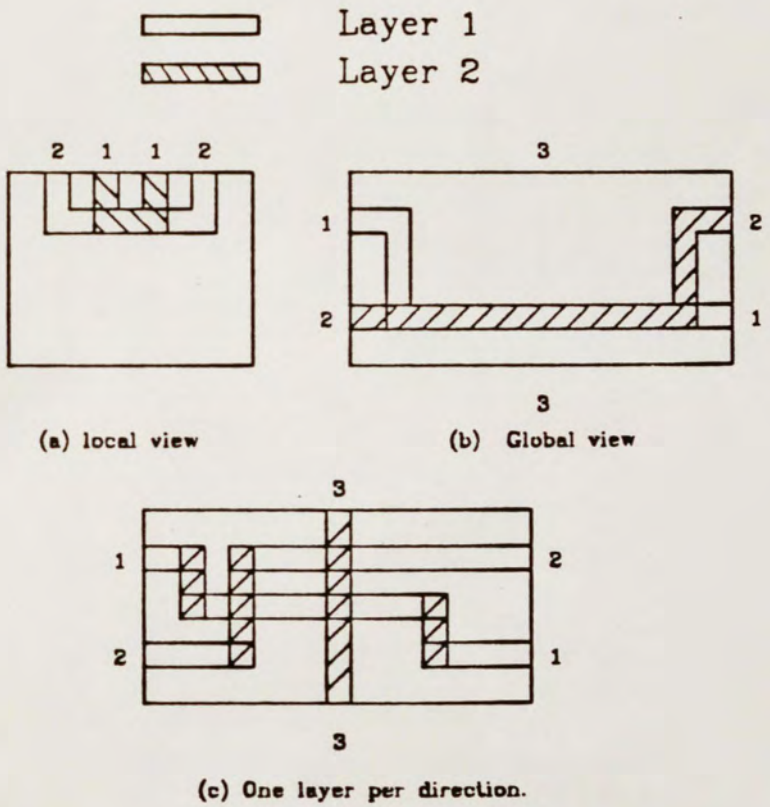


Figure 4.4. Layer assignment examples.

have a dramatic effect on the routing of future nets. In Figure 4.4(b), Nets 1 and 2 have been routed using only one horizontal track. However, since only two routing layers are available, this choice prevents net 3 from being routed. Also, overlapped nets are susceptible to capacitive coupling between the wires, an undesirable electrical phenomenon. Figure 4.4(c) illustrates a solution using required directions for each routing layer. This layer assignment minimizes the impact on future nets and is used on most sequential (and parallel) routing algorithms.

4.3.2. Pin Blocking

Even using a single routing layer per direction, the path chosen for a net can still dramatically affect the ability to route future nets. Figure 4.5 gives a typical example. Here the path chosen for net 1 prevents net 2 from gaining access to the routing region. This problem is called *pin blocking*.

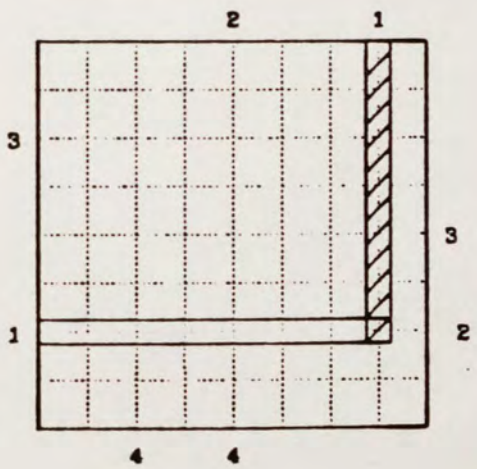


Figure 4.5. Example of pin blocking

The original Lee algorithm choses paths blindly, with little or no regard for nets yet to be routed. Smith analyzed the completion rates for the Lee and his results are reproduced in Figure 4.6. Because uniform pin distributions around the periphery of the routing region are the most representative of integrated circuit routing, and they also appear to be the most difficult, Smith choose this pin distribution for comparison of the algorithms presented. The completion rate of the Lee drops off dramatically when the MAM reaches .1. Recall the very sparse region in Figure 4.3 had a MAM of .11, indicating that the Lee has very poor performance.

Two techniques have been used to decrease the effects of pin blocking. The first is called *net ordering*. In Figure 4.5, had net 2 been routed before net 1, the pin blocking problem would not have

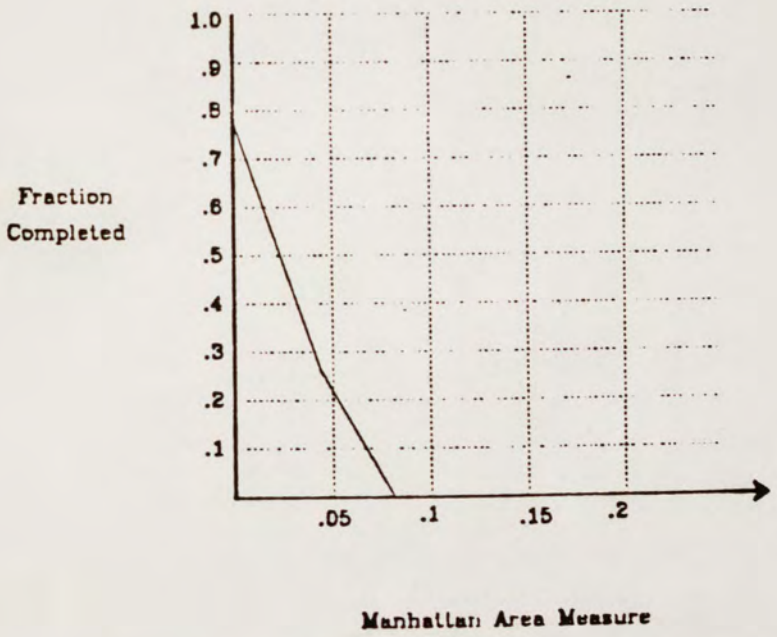


Figure 4.6. Completion rate of the Lee.

occurred. In general, however, net ordering is difficult to effectively implement and it is not difficult to suggest examples that do not have any preferred order, i.e., any net ordering choice will result in a pin being blocked.

A more direct approach is suggested by Saxe and reported by Smith. His technique places tails on each pin extending across 50% of the routing region. These tails behave much the same as wires and are called *reservations*. During the routing process, only wires that belong to the same net are allowed to intersect the reservation. The use of reservations is intended to guarantee each pin access to the routing region. Once a net has been completed, any unused section of the reservation is returned to general use. Figure 4.7(a) illustrates the initial reservations and Figure 4.7(b) shows the results after routing net 1. Note that the reservation for pin 2 made it impossible for net 1 to block net 2's access to the routing region.

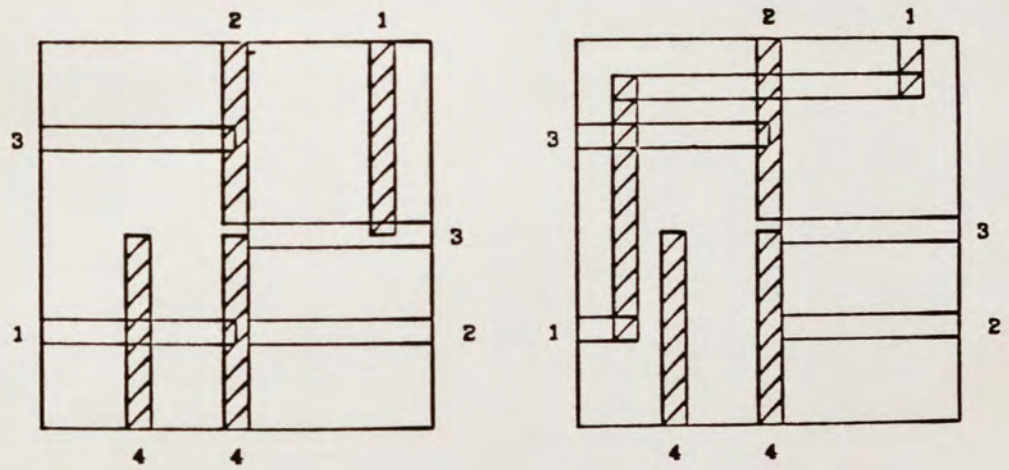


Figure 4.7. Example using reservations

Smith showed that using reservations in conjunction with the Lee increased completion rates by almost an order of magnitude. His results are reproduced in Figure 4.8 and compared to the Lee.

4.3.3. Vias

Vias can cause problems similar to the layer assignment problem. Vias form blockages in the routing area. These blockages present obstructions to future paths. Consider Figure 4.9. Here, two possible paths are shown for net 1. If the diagonal path is chosen, the vias will prevent net 2 from being completed.

Agrawal[Ag73] studied the effect of blockages on a square routing region by randomly placing obstructions over the routing region and measuring the probability of being able to route a random two-point net. His study indicated that as the percentage of blockages reached

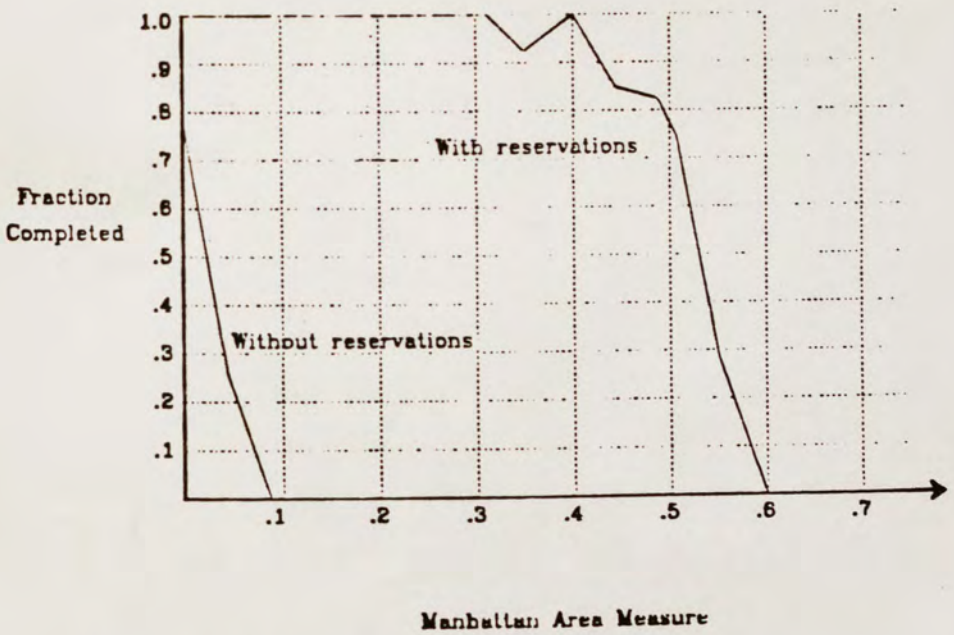


Figure 4.8. Completion rate of the Lee with Reservations

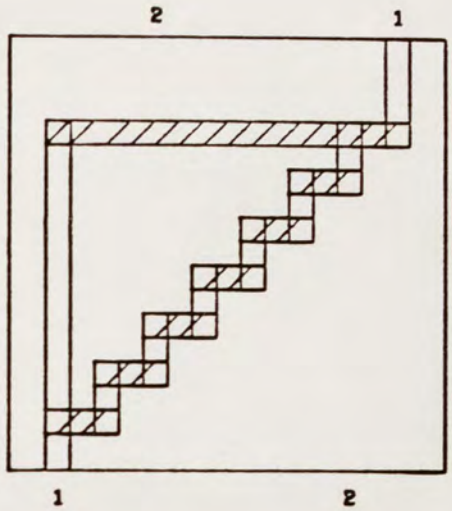


Figure 4.9. The problem with vias.

35%, the chances of being able to route a future net fell off dramatically. His experiments were also run on blockages that were created by the Lee algorithm. His statistics indicated that the vias created by the Lee caused the ability to route future nets to fall off faster than the rate for randomly distributed blockages. He hypothesized that this was due to the Lee creating higher local densities.

One technique aimed at reducing the number of vias is to utilize the graph routing region model used by the variable cost Lee router (Chapter 2.1). If the weight assigned to a via edge is greater than the longest path in either of the routing layers, the shortest path algorithm will always find the path with the minimum number of vias. This technique will route each net individually using the minimum

number of vias, but does not attempt to minimize the overall number of vias.

Consider Figure 4.10. This solution required a total of 9 vias. Had net 2 been routed in a way that did not interfere with net 3 or 4, all three nets could have been completed using only 5 vias.

One way to achieve this result would be to route nets 3 and 4 before routing net 2. Changing the order that nets are routed is called *net ordering*.

4.4. Degree of Freedom Based Routing

In this section a net ordering scheme is developed based on the number of possible path choices for a net. This routing algorithm is functionally equivalent to the LUZ algorithm, but is implemented using a variable cost Lee algorithm. The result is a router with higher completion rates, that is extensible to multipoint nets, and has the advantage over the LUZ that it halts.

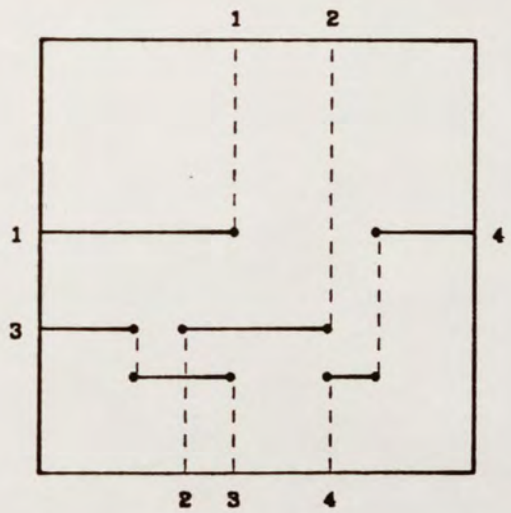


Figure 4.10. Routing via interaction

4.4.1. Systems of Distinct Representatives

Consider the following subproblem. Let E be a set and $A_1, A_2, \ldots, A_n$ be a family of subsets of E. A family $e_1, e_2, \ldots, e_n$ of elements of E is a *system of representatives* of $A_1, A_2, \ldots, A_n$ provided e_1 is in A_1, e_2 is in $A_2, \ldots,$ and e_n is in A_n. If, in addition, $e_1, e_2, \ldots, e_n$ are distinct elements, then $e_1, e_2, \ldots, e_n$ is a *system of distinct representatives* for $A_1, A_2, \ldots, A_n$.

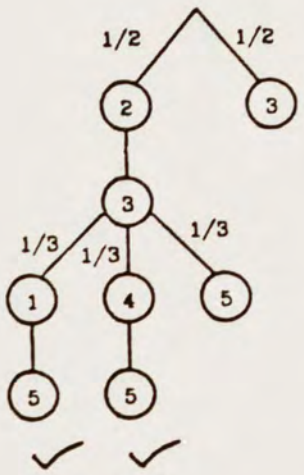
Consider the example shown in Figure 4.11. A system of representatives might be 2,3,3,5. A system of distinct representatives might be 2,3,4,5.

Assuming that there is a system of distinct representatives for the set E, what is the probability of randomly selecting a system of distinct representatives using only the previous selections to guide the selection process?

This probability can be computed for the example in Figure 4.11 by constructing the decision tree shown in Figure 4.12. As the tree

$$
\begin{aligned}
E &= \{\, 1,2,3,4,5 \,\} \\
A_1 &= \{\, 2,3 \,\} \\
A_2 &= \{\, 3 \,\} \\
A_3 &= \{\, 1,4,3,5 \,\} \\
A_4 &= \{\, 5 \,\}
\end{aligned}
$$

Figure 4.11. Example of systems of distinct representatives.



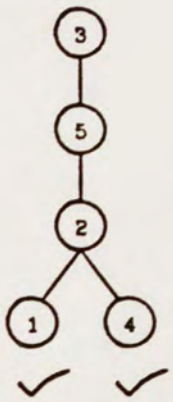
$$P_{ran} = \frac{1}{2} \cdot \left(\frac{1}{3} + \frac{1}{3} \right)$$
$$= \frac{1}{3}$$

Figure 4.12. Decision tree using random set ordering

indicates, either 2 or 3 can be selected from set A_1. If 3 is chosen, 3 cannot be selected from A_2 since this element was selected in A_1. This path will terminate in failure. If 2 is chosen from A_1, 3 must be selected from A_2. Choices for A_3 have now been reduced to {1,4,5}, because 3 has already been selected from A_3. This process continues for the rest of the tree. Successful paths are marked with an check. The probability of successfully finding a set of distinct representatives can be computed using the probability of making a correct choice at each level in the tree. Figure 4.12 computes this probability at 1/3.

Assume the ordering is changed so that elements are selected from set A_2, A_4, A_1 and A_3. The decision tree for this ordering is shown in Figure 4.13. The probability of finding a system of distinct representatives in this case is 1. The dramatic improvement is clearly related to the ordering chosen. Why was this ordering selected?

Let us define the degree of freedom for a set A_i, denoted dof_{A_i}, to be the number of elements minus 1 in the set. For example, in Figure



P = 1

Figure 4.13. Decision tree using *dof* ordering.

4.12, $dof_{A_1}=1, dof_{A_2}=0, dof_{A_3}=3$ and $dof_{A_4}=0$. The degree of freedom is a measure of the flexibility involved in the selection. The order that sets were selected in Figure 4.12 was random, while in Figure 4.13, the selection was based on increasing degrees of freedom. This strategy postpones the selection of items having more flexibility, decreasing the chances that an incorrect selection is made. Although this strategy will not always find a system of distinct representatives, it clearly is as good or better (usually much better) than the random case.

4.4.2. Degree of Freedom (DOF) Algorithm

This notion of degree of freedom is useful in the area routing problem. Let the position of the two pins in an arbitrary two point net determine the *intrinsic* degree of freedom according to the following table:

dof	Condition
0	if pins are directly opposite one another
0	if pins are on adjacent sides
1	if pins are on opposite sides

The degree of freedom in these cases is equivalent to the number of edges that are not *fixed* by the position of the pins. For example, a Z shaped path has the two endpoints fixed but the middle segment is free to move, creating the one degree of freedom.

Now return to the example in Figure 4.10, reproduced for convenience as Figure 4.14. The original order chosen to route the nets was based only on the net number. In Figure 4.14, the nets are routed according to increasing degrees of freedom. Thus, the degree of freedom of net 1, denoted dof_1, is 0. The dof_2=1, dof_3=0, and dof_4=0. The result is a routing that uses only 5 vias, reducing the number of blockages in the region and increasing the chances that subsequent nets can be completed.

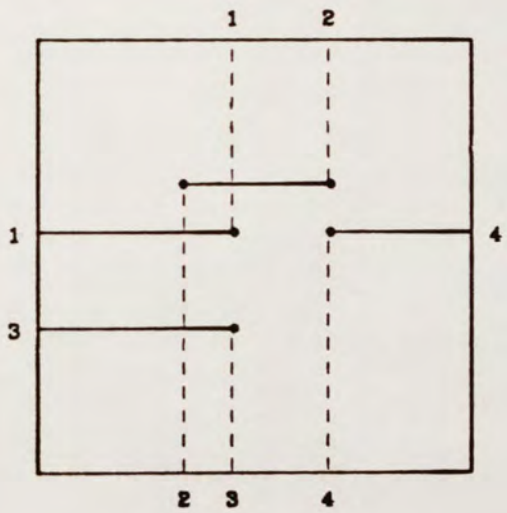


Figure 4.14. DOF routing example

This degree of freedom analysis also provides a clue as to what should be done if a net cannot be completed using the intrinsic number of vias.

Given a path with n vias, the *dof* is dynamically defined as

$$\begin{aligned} dof &= 0 \quad \text{if } n = 0 \\ dof &= n\text{-}1 \quad \text{if } n > 0 \end{aligned}$$

Consider Figure 4.15. In this example, all three paths have an intrinsic degree of freedom 0. When two or more nets have the same degree of freedom, the routing order is arbitrary. Net 1 can be routed using only one via. Net 2 now cannot be routed using one via because of a conflict with net 1. At this point it is clear that Net 2 must be routed using a path with at least 3 vias. This indicates that the new path has *dof* equal to 2. Hence, the list of nets yet to be routed is dynamically updated with net 2 placed after all nets with less than 2 degrees of freedom. Thus net 3 would be routed, followed by net 2.

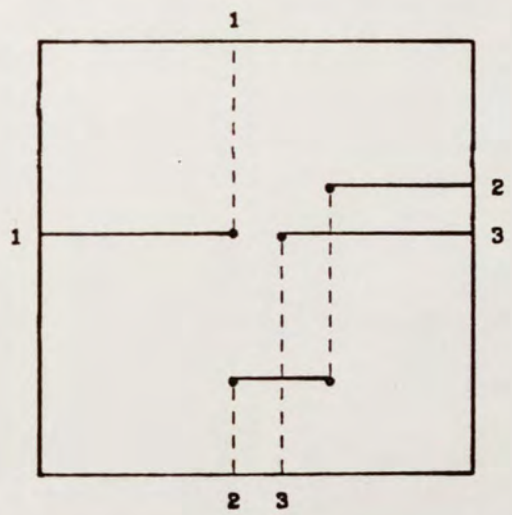


Figure 4.15. Dynamic modification of net *dof*.

The following algorithm implements Degree Of Freedom, DOF, based routing.

Degree Of Freedom algorithm

```
1       i = 0;
2       while( # nets remaining != 0) {
3          for each(net not routed) {
4                   route net using no more than i vias.
5                   if success mark the net as finished and
6                     decrease the number of nets remaining.
7                   }
8          i = i + 1;
9          }
```

For each iteration of the while loop, each unrouted net is attempted using no more than i vias. Remaining nets are attempted until all the nets have been completed.

The routing of each net is accomplished using the variable cost Lee algorithm presented in Chapter 2.1. Let V be greater than the longest path in a single layer. The cost of a via edge in the variable cost Lee is set to V. Thus, the maximum cost of a path with n vias is $((n+1)*V)-1$. Thus, to restrict the path found by the Lee to less than i vias, the standard Lee algorithm is run, but the Lee does not allow any node to be placed in the cost list that has a weight greater than $((i+1)*L)-1$. By not allowing these nodes to be placed in the cost list, those paths that contain more than i vias are "pruned" from the search path. *Pruning* simply means that a possible path was not considered because the cost of such a path exceeded the number of vias allowed in this pass. For example, on the first pass (i = 0), only paths with zero vias are attempted.

Ordering the nets in this manner is functionally equivalent to routing them using the LUZ algorithm presented earlier, with the added advantages of being easy to implement and understand. The

DOF router also has the advantage of knowing when to halt. In the original LUZ, the algorithm must try successively more complex paths, although Smith's studies indicate that little or no improvement occurs beyond LUZ level 3 paths. The Lee based approach can determine if any more paths are possible using the next degree of freedom by recording whether any paths were "pruned" by the current degree of freedom. If paths have been pruned, other paths of more complexity have been overlooked. If no paths were "pruned" and the Lee failed to find a path, no increase in the number of vias will help; all paths are blocked.

Finding improved implementations of the LUZ algorithm is important because the LUZ algorithm was shown by Smith to be have higher completion rates than the Lee. His results are shown in the graph in Figure 4.16. In light of the discussion concerning degree of

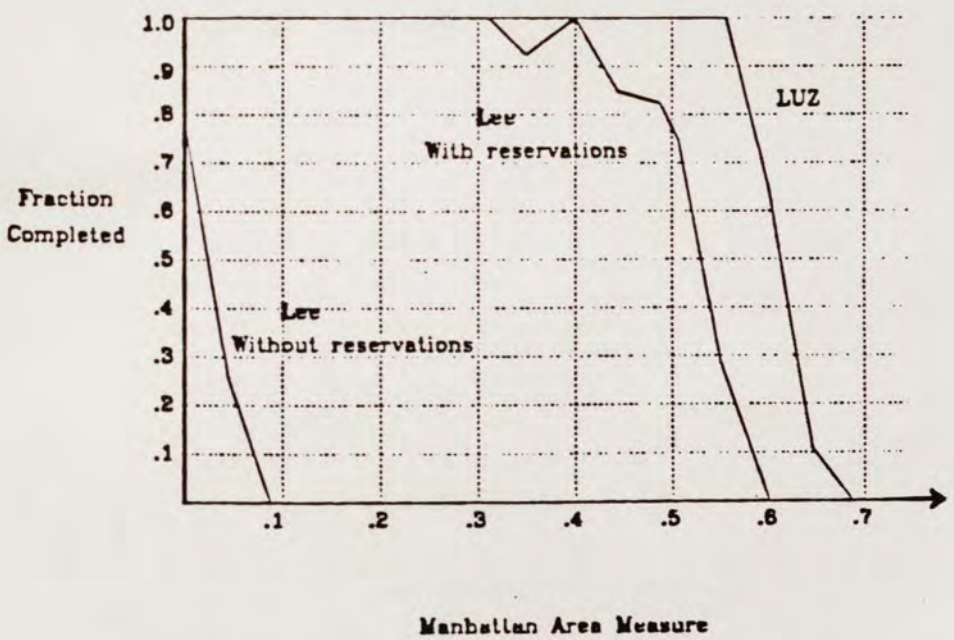


Figure 4.16. Comparison of the Lee and LUZ algorithms.

freedom routing, these results should not be surprising. The Lee is capable of selecting the same path shapes that are used by the LUZ. The difference in completion rates is due to the net ordering imposed by the LUZ. More complex paths are postponed until simpler paths have been completed. Thus, we would expect the Lee based DOF algorithm to perform no worse and probably better than the Lee. Smith's comparison of the Lee and LUZ algorithms was restricted to square routing regions consisting of only two point nets. Later in this chapter, the DOF algorithm is extended to consider multipoint nets. The MAM is also extended to provide a lower bound on the number of grid points required by multipoint nets. In Chapter 6, extensive comparisons of the DOF and Lee algorithms are made over a variety of sample areas.

The DOF algorithm has interesting performance characteristics. Since the cost values in the wavefront are bounded, the number of cells considered at each stage will be a function of the number of cells reachable using less than or equal to i vias. Because the path complexity increases with each pass, the run time for each successive path should be longer. At the same time, the number of unused grid points in the routing region is decreasing, and thus the search time for a path decreases. Smith indirectly provided an upper bound on the execution time of the DOF algorithm. His results indicated that routing paths with complexity over LUZ level 3 (≤ 6 *vias*) had little or no effect on completion rates. Since each pass through the DOF algorithm allows one additional via in the path, we would expect to make less than 6 passes through the algorithm before halting.

We can provide a speedup of two to the DOF algorithm by partitioning the nets into two groups based on intrinsic degree of

freedom. If a net has an intrinsic $dof = 0$, the number of vias in the path must be odd. Likewise, if the intrinsic $dof = 1$, the number of vias in the path must be even. Thus, if i is even, the Lee need only be run on nets with intrinsic $dof = 1$. Likewise, if i is odd, the Lee need only run on nets with intrinsic $dof = 0$.

Using Smith's result and the improvements above, approximately six passes should be made through the algorithm, with alternate passes using a partition of the nets remaining. Thus, we would expect that the performance of the DOF algorithm would be no more than three times the Lee. In practice, the performace on the DOF algorithm was approximately 1.8 times that of the Lee.

4.5. Extending Degree of Freedom Routing to Multipoint Nets

The LUZ algorithm proposed by Smith cannot be used on multipoint nets. Since most area routing problems contain multipoint nets, this severely restricts the usefulness of the LUZ. This section will identify the problems created by multipoint nets and propose the revisions necessary to extend the DOF algorithm to these cases.

The multipoint net problem is equivalent to the geometric Steiner tree problem[Ga79].

Problem: Geometric Steiner Tree Problem
Instance:
Set $P \subseteq Z \times Z$ of points in the plane,
positive integer k.
Question:
Is there a finite set $Q \subseteq Z \times Z$ such that
there is a spanning tree of total weight
k or less for the vertex set $P \cup Q$,
where the weight of an edge $\{(x_1,y_1),(x_2,y_2)\}$
is the rectilinear metric $|x_1-x_2| + |y_1-y_2|$.
A spanning tree that minimizes the value of k is called a
Minimum Weight Steiner Tree (MWST).

Consider Figure 4.17(a). The set P is called the set of *distinguished*

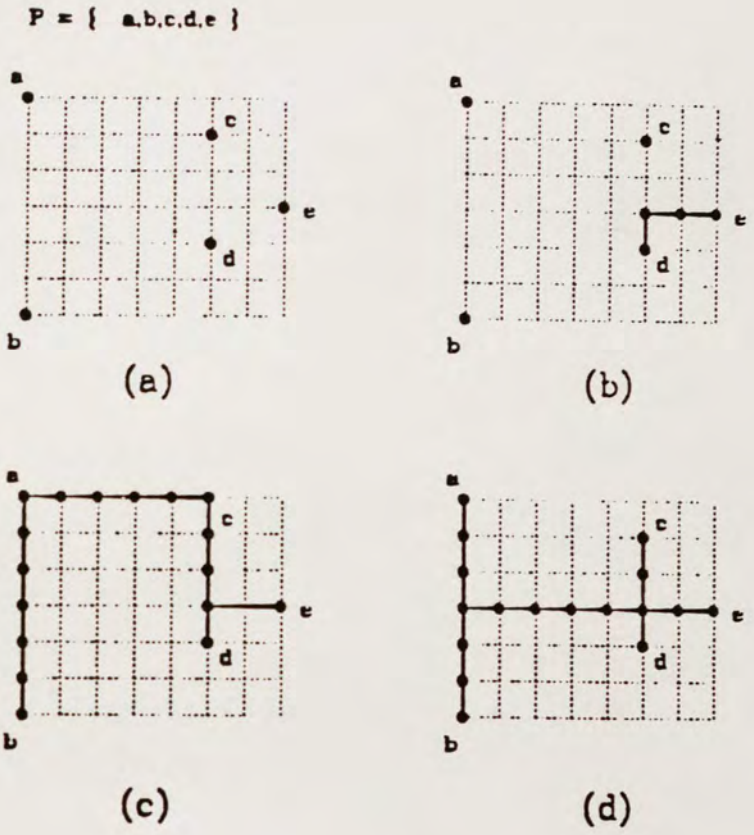


Figure 4.17. Steiner tree example.

vertices, or vertices that must be included in the Steiner tree's vertex set. In routing problems P represents pins in a signal net and k represents the minimum amount of wire necessary to complete the net. Garey and Johnson [GaJo79] showed this problem to be NP-complete. This indicates that finding a *MWST* is very difficult. Fortunately, good heuristic algorithms are well known.

Figure 4.17(d) illustrates a *MWST* for Figure 4.17(a). The points in P are connected using a minimum number of edges. A popular heuristic commonly used in conjunction with the Lee makes use of Dijkstra's shortest path algorithm to produce a Steiner tree. The

algorithm begins by finding the shortest rectilinear distance between any two pins (elements of P). In Figure 4.14(a), the two elements of P with the shortest rectilinear distance are d and e, or c and d. If two or more sets of points have equal distance, the choice is arbitrary. These points are connected using the path that produced the shortest distance. As shown in Figure 4.17(b), d and e are chosen to be connected. This partial net is called a *subnet*. Then, the closest distinguished vertex to any point in the subnet is chosen, and this path is added, creating a larger subnet. In the example, c has the minimum distance (2) from the subnet. This connection is made. This process is repeated until the entire net has been completed. A completed Steiner tree using this heuristic is shown in Figure 4.17(c). The minimum weight Steiner tree is shown for comparison in Figure 4.17(d). Notice that the cost of this Steiner tree is 17, as opposed to the minimum weight Steiner tree's weight of 16.

Baratz[Ba81] showed that the weight of a Steiner tree generated using this algorithm is no more than twice the optimal weight.

A variation on this heuristic is commonly implemented using the Lee algorithm. The first pin in a net is selected as the source and all other pins in the net are labeled as targets. The Lee forms a subnet by connecting the source to the closest pin. The Lee then arbitrarily chooses another pin in the net and uses it as the source and the subnet as a destination. This process continues until the entire net is completed.

The DOF algorithm presented in the previous section is based on the Lee algorithm. Using the techniques explained above, the Lee based DOF algorithm can also implement the Steiner tree heuristic. During each pass of the DOF algorithm, connections are attempted

between pins of the same net using paths with i vias. If the connection between a pin and the subnet cannot be accomplished using only i vias, the net is marked as incomplete, and the completion of the net is postponed until the next pass.

4.6. Extending the Definition of MAM to Multipoint Nets

The MAM, presented in section 4.2, is used as a measure of the complexity of the routing region. This measure is used to compare the completion rates of various algorithms and to determine the size of the routing areas that must be provided during the placement of blocks in the general routing problem. This section extends the notion of MAM to multipoint nets.

Recall from section 4.2 that the MAM provides a lower bound on the number of grid points necessary to complete the wiring. If all edges are of weight one, the the weight of the *MWST* is equal to the number of edges in the tree. The number of vertices in a tree is always equal to the number of edges in the tree plus one. Thus, the *MWST* also minimizes the number of grid points. In this section we will develop a reasonable lower bound on the weight of the *MWST*. This lower bound is shown to be equal to the weight of the *MWST* for two and three pin nets. Since two and three pin nets make up the vast majority of all nets in custom routing, this measure provides an excellent lower bound on the the minimum number of edges in the *MWST* and hence the number of grid points necessary to complete the wiring.

Assume we are given an instance of the rectilinear Steiner tree problem. Let $P=\{p_1,p_2, \ldots ,p_m\}$ be the set of distinguished vertices where $p_i=(x_i,y_i)$. In the area routing problem, p_i is the closest

adjacent point to pin i. This distinction merely implies that p_i is in the routing region while pin i is connected to the bounding rectangle. If $m = 2$, the *MWST* is formed by the minimum length path between p_1 and p_2. The weight of this tree is the rectilinear distance between the two points: $|x_1-x_2|+|y_1-y_2|$. We define the bounding rectangle $R(p_1,p_2,\ldots,p_i)$ as the minimum rectangle that encloses vertices $p_1,p_2,\ldots,p_i)$.

Lemma 4.1: For each additional vertex, p_i for $2>i\geq m$, the additional weight of the MWST is greater than or equal to the distance from p_i to $R(p_1,p_2,\ldots,p_{i-1})$.

Proof: First we observe that a MWST on i vertices has exactly one path from $R(p_1,p_2,\ldots,p_{i-1})$ to p_i. Clearly multiple paths would have a weight greater than simply connecting the two exit points using a path formed by the perimeter of R and connecting p_i to the closest point in this path. Thus, the additional weight of the MWST is greater than or equal to distance from p_i to the exit point of the path.

Thus, to compute a lower bound on the weight of the *MWST*, two points of the net are chosen and the manhattan distance between them is computed. For each additional point, A distance is computed to the bounding rectangle of the previous points and is added to to the previous sum. The total provides a lower bound on the number of edges in the *MWST*. Figure 4.18 gives an example. P initially contains $\{p_1=(0,3),p_2=(5,0),p_3=(3,5),p_4=(10,3)\}$. The manhattan distance between p_1 and p_2 is 8. Thus 8 is the lower bound on the weight of the *MWST* for the first two points. Lemma 5.1 guarantees that the weight of the *MWST* for the first three points is greater than the weight of the *MWST*

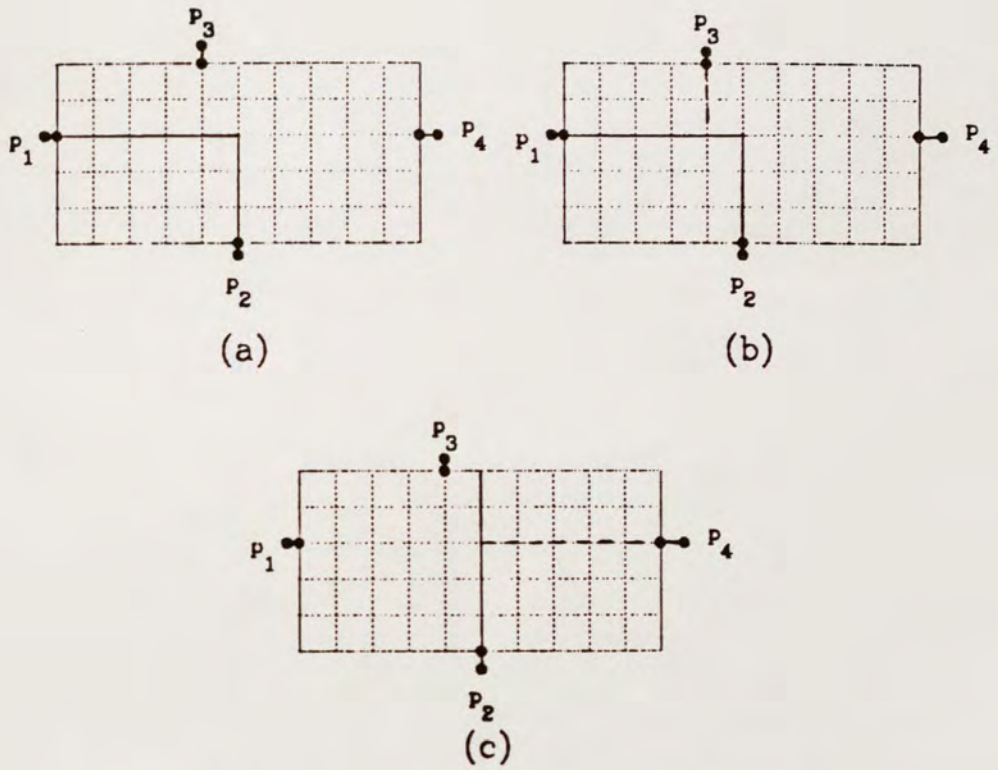


Figure 4.18. MWST lower bound example.

for the first two points plus the manhattan distance to the smallest rectangle that encloses the first two points. Figure 4.18(b) illustrates the distance for the addition of p_3. Thus, a lower bound on the weight of the *MWST* for the first 3 points is 10. Figure 4.18(c) shows the distance required by the addition of the last point, p_4. A lower bound on the weight of the *MWST* is 15. The minimum number of grid points necessary to complete the wiring is then equal to the weight of the *MWST* plus one or 16.

The proof that the lower bound is tight for 3 points is shown in Lemma 4.2.

Consider any three points, P, where $P=\{p_1,p_2,p_3\}$.

Lemma 4.2: The *MWST* for P is equal to the manhattan distance from p_1 to p_2 plus the manhattan distace from (x_3,y_3) to the closest point in $R(p_1,p_2)$.

Proof: Let p_4 be the closest point in $R(p_1,p_2)$ from p_3. Clearly the *MWST* for p_1 and p_2 can be formed in such a way as to include p_4. Thus, the minimum path between p_3 and P_4 creates a tree on three vertices. Lemma 4.1 guarantees that the weight of this tree is less than or equal to the weight of the *MWST*.

4.7. Extending the MAM to the One Layer Per Direction Wiring Model

The MAM proposed by Smith does not make any assumptions concerning the wiring model used. For example in Figure 4.19, the L shaped path will use a minimum of 5 grid points. The MAM for this net will then be $\frac{5}{50}$. If we restrict the wiring model to one layer in the vertical direction and one layer in the horizontal direction, a via must be used to swtich from one layer to the other This via will use both a

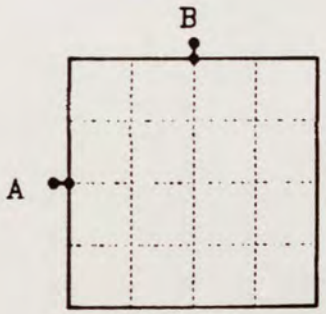


Figure 4.19. Extended MAM model.

grid point on the top layer and a grid point on the bottom layer. Thus, the minimum number of grid points for the net is 6 and the MAM is $\frac{6}{50}$. Clearly, this revised definition of the MAM provides a tighter bound on the number of grid points necessary to complete the wiring.

The value of the MAM for this wiring model is easily calculated as the manhattan distance plus one (the old value of the MAM) plus the minimum number of vias in the path. Since the algorithms presented use the one layer per direction wiring model, the difficulty of the routing region will be exclusively measured using this definition of the MAM.

4.8. Summary

This chapter described the area routing problem and its importance in routing custom integrated circuits. Smith's MAM was introduced as a means of measuring the complexity of an area. The chapter went on to describe characteristics of area routing problems such as layer assignment, pin blocking and vias. The LUZ concept of "simplest" paths was explained in terms of systems of distinct representatives. This new definition enabled us to develop a functionally equivalent definition of the LUZ algorithm that was extensible to multipoint nets and halted if a path could not be found for a given net. The notion of MAM was also extended to measure the complexity of areas that contain multipoint nets and to utilize the one layer per direction routing model. In Chapter 6, these extensions allow a comparison of completion rates using the Lee and DOF algorithms on areas with multipoint nets. The results will show that the DOF algorithm has better completion rates than the Lee algorithm

over a variety of sample channels using both two point and multipoint nets.

CHAPTER 5

Measurements of Algorithm Performance over Channels

A *channel* was previously defined as a routing region with fixed terminal positions on the top and bottom of the region and "floating" connections on the right and left ends. In gate array, polycell and standard cell layouts, channels are naturally formed between rows of logic elements. An *area* was defined as a routing region with fixed pins on other than the top and bottom sides of the region. In custom integrated circuit layouts, some regions form natural channels while others require an area routing solution.

This chapter compares the solutions generated by both channel routing algorithms and area routing algorithms using a variety of randomly generated channels. The randomly generated channels are created using both an algorithm developed by Rivest and one developed by the author. The channel routing algorithms used in this comparison are the Dogleg, Revised Dogleg and LCP algorithms that were described in Chapter 3. The area routing algorithms used are the Lee and DOF algorithms. The area routing algorithms are extended in this chapter to incorporate the additional flexibility of floating endpins.

The comparison of channel routing solutions and area routing solutions over channels is important for the following reasons:

(1) Channel routing algorithms have different strengths and weaknesses from area routing algorithms. This chapter determines which characteristics are most important in achieving an optimum number of tracks in the channel.

(2) Area routing algorithms are used to route custom integrated circuits. Some portions of these circuits form natural channels. If the area routing algorithms have low completion rates in randomly generated channels, we would expect similar results when these algorithms are applied to the custom routing problem. This problem is especially acute in routing projects such as the PI system. Here, natural channels are turned into areas with fixed pins on all four sides of the region. Thus, the flexibility of floating endpins is taken away. If the Lee and DOF algorithms perform poorly on channels with floating endpins, we expect that the reduced flexibility imposed by the PI project would further reduce the algorithm's completion rates.

The results of this section will clearly indicate that the Revised Dogleg and LCP algorithms, developed in Chapter 3, use significantly fewer tracks over the density than the Greedy algorithm, which uses fewer tracks than the Dogleg. The results also show that area routing algorithm's use significantly more tracks over the density than any of the channel routing algorithms.

5.1. Algorithm Characteristics

Each algorithm used in this comparison has particular strengths and weaknesses. Three characteristics that are of particular interest are horizontal line segment organization, vertical constraints and path shapes.

Horizontal line segment organization refers to how well an algorithm minimizes the amount of white space between two different segments. The Dogleg does an excellent job of minimizing this white space. The Greedy router also does an excellent job of aligning

horizontal line segments. At each column, the Greedy router brings a new net into the first available track. This process tends to pack line segments together. Conversely, the DOF and Lee algorithms make no effort to minimize the amount of white space between segments. The Revised and LCP algorithms are loosely based on the Dogleg algorithm. These algorithms, while minimizing white space, also chose segments in ways that maximize density reductions.

In terms of vertical constraints, the Dogleg based algorithms do an excellent job of considering vertical constraints. The Greedy router indirectly handles vertical constraints by extending each terminal to the nearest available track and adding the wiring necessary to reduce the distance between the track a net is currently running in and the corresponding pin. The use of reservations in the Lee and the DOF algorithms implements a very basic form of vertical constraint.

In terms of path shapes, the Lee and DOF algorithms have the advantage. Each is capable of using arbitrarily shaped paths to complete a net, although the net ordering implied by the DOF algorithm tends to use complex path shapes more wisely than the Lee. The Dogleg, Revised Dogleg and LCP algorithms are forced to use a very restricted set of path shapes. The Greedy router uses more complex path shapes than the Dogleg but not as complex as those offered by the Lee and DOF algorithms.

Since each algorithm has particular strengths and weaknesses, this comparison will also help determine which characteristics are most important in achieving high completion rates.

5.2. Algorithm Normalization

As we have seen in Chapter 3, the input to the channel routing problem is shown in Figure 5.1(a). Each position on the top and bottom of the channel has either the net number of the terminal at that position or a blank, indicating that no terminal is located at that position. On the left and right sides of the channel, net numbers indicate that the net must connect to some point on that side of the channel. A solution to the channel routing problem is shown in Figure 5.1(b). The objective of each algorithm is to minimize the number of horizontal tracks necessary to complete the interconnection of the nets.

This section will discuss modifications in the Greedy, DOF and Lee algorithms necessary to produce comparable solutions.

5.2.1. Eliminating Extra Columns Generated by the Greedy

The Greedy router, as described by Fiduccia and Rivest, generates routings that can use more columns than were given in the original problem description. In Figure 5.2 the number of columns, n, specified in the problem is 20. The solution produced by the Greedy

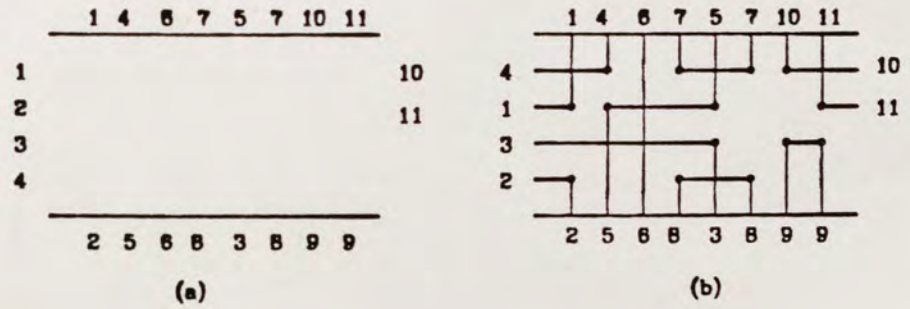


Figure 5.1. Channel routing problem

router used 21 columns. Since this additional area is not used by the Dogleg, Lee or DOF algorithms, a fair comparison of the algorithms requires the elimination of these extra columns.

Let k be the number of horizontal tracks used in a solution and the *density*, D, be the minimum number of tracks necessary to complete the routing.

The greedy router takes an initial value for k as an input parameter and increases k as necessary to complete the routing. Our implementation sets the initial value of k to D. If the completed channel uses more than the number of columns specified in the problem description, the algorithm is rerun, increasing the initial value of k by one. This process continues until no net extends beyond the n columns specified in the problem statement.

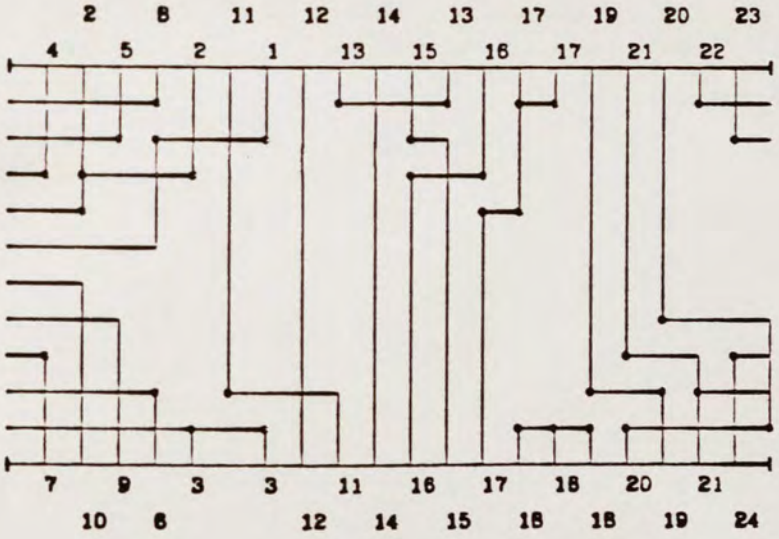


Figure 5.2. Extra column example

This method of extra column elimination, although effective, is rarely necessary. Over the hundreds of test cases applied to the Greedy router, less that 2% required greater than n columns.

5.2.2. Minimizing the Routing Area using the Lee or DOF

In the Dogleg and Greedy routers, k is automatically increased from D as necessary to ensure completion of the channel. In contrast, the Lee and DOF algorithms are given a fixed area and either produce a valid routing or return failure, indicating that the algorithm did not produce a solution in the given area. To find the minimum area required for the completion of a given channel using the Lee and DOF algorithms, these algorithms are first run using $k=D$ tracks. If the routing fails, k is increased and the algorithm is rerun. This process continues until the channel is successfully completed.

5.2.3. Extending the Lee and DOF with Floating Endpins

The Dogleg and Greedy algorithms are given the flexibility to choose the positions of pins on the left and right edges of the channel. The Lee and DOF algorithms, however, require that the positions of all pins be fixed. Fixed pin positions take flexibility away from the Lee and the DOF.

To complete a fair comparison, the Lee and DOF algorithms are extended to choose the positions of floating endpins. Figure 5.3 shows a two point net to be routed. Pins A and B belong to the same net. The position of pin A is fixed while the position of pin B is selected by the algorithm. The extended Lee and DOF algorithms use Pin A as the source and all unused locations on the side of pin B as the target. These positions are marked in Figure 5.3(a). Using these selections

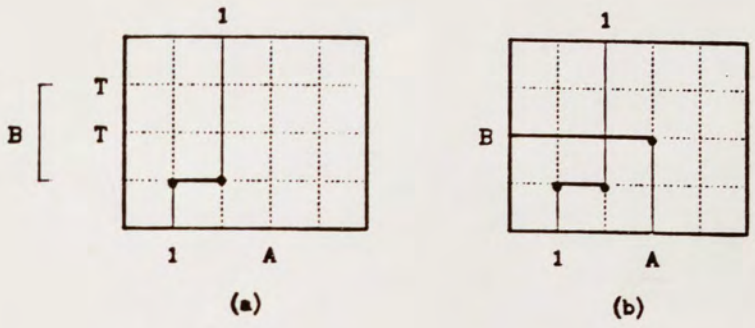


Figure 5.3. Floating pin extension.

the first target pin reached is used as the location of pin B. Figure 5.3(b) shows the final position of B and the path selected by the Lee or DOF.

This extension is consistent with the Lee's performance in custom layouts. If a channel is present in the routing region, the Lee chooses the crossing point along the edge of the channel using a path that does not conflict with earlier crossings.

5.3. Test Channel Generation

To verify each routing algorithm's overall effectiveness, this section describes algorithms that generate sample channels. Each algorithm is parameterized to create sets of channels exhibiting particular characteristics.

5.3.1. Rivest's Channel Generation Algorithm

The first algorithm was written in LISP[Wm81] at MIT by R. Rivest[Ri82b]. The algorithm was rewritten in "C"[KeRi78] by the author for use in these experiments.

The algorithm accepts the following parameters:

n	The channel width (number of vertical columns)
D	The channel density
c	The congestion of the channel (i.e. what fraction of the terminal positions are actually used along the top and bottom of the channel)
u	The number of nets that must enter from the left end of the channel and exit from the right. Our implementation fixes $u=0$.
t	The approximate number of terminals per net If $t=2$ each net is restricted to exactly 2 terminals.
a	If a = true the vertical constraint graph is acyclic.
r_0	The initial random number seed.

A typical example of a channel generated by Rivest's algorithm and routed by the Dogleg is shown in Figure 5.4 using the following input parameters:

Rivest(n=20, D=10, t=2.5, c=.9, u=0, a=true)

Rivest's algorithm begins by generating a set of nets, L, that enter the channel from the left end, where $L=\{1,2,\ldots,D\}$ and D is the channel

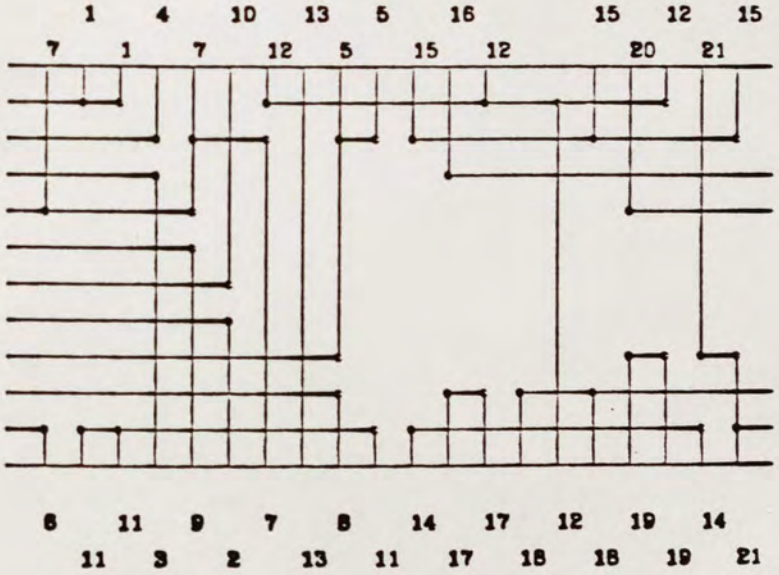


Figure 5.4. Sample channel using Rivest's algorithm

density input parameter. A set N, the currently active nets, is created with $N=L$. For a given column, the set N contains the nets that enter the column from the left. For each column i from 1 to n, the algorithm determines, based on the value of c, whether to place terminals on the top and/or bottom of column i. Terminals are either chosen from N or a new net is created, in which case the new net is added to N. To ensure that the number of nets in N does not exceed the density, a new net is never created if $|N|=D$. Terminals chosen from the set of currently active nets are removed from N with probability $\frac{1}{(t-1)}$. If t is equal to 2.0, nets are automatically removed from the current set as soon as the first two pins are placed. If t is greater than 2.0, the number of pins per net is exponentially distributed. In practice, the average number of pins per net is less than t. For example in these test cases, the average number of pins per net if $t=2.5$ was 2.39. After all columns have been considered, the remaining elements of N are placed in R, the set of nets that enter the channel from the right.

5.3.2. Alternate Channel Generation Algorithm

Rivest's algorithm generates channels that have several properties that are not generally found in custom layouts.

(1) The left edge of the channel always is at the density with the right edge usually less than the density. This causes asymmetric channels.

(2) The left to right approach also tends to group connections for a net. If a new net is created in column i, the probability of the net being terminated before column $i+s$ increases as s increases.

The Alternate channel generation algorithm generates channels which do not exhibit these characteristics. The algorithm takes 5 input parameters and a random seed:

n	The channel width (number of vertical columns)
c	The congestion of the channel (i.e. what fraction of the terminal positions are actually used along the top and bottom of the channel)
l	The channel height (approximately $c^* l$ endpin positions are used as terminals)
t	The approximate number of terminals per net. If $t=2$ each net is restricted to exactly 2 terminals.
a	If a = true the vertical constraint graph is acyclic.
r_0	The initial random number seed.

It is important to note that these parameters do not include D, the channel density. The alternate algorithm makes no attempt to control the density of the channels produced. Figure 5.5 shows an example of a channel generated by this algorithm using the parameters

Alternate (n=30, c=.9, l=10, t=2.5, a=true)

The algorithm begins by defining a rectangular area with height $l+1$ and width $n+1$. The algorithm places the first and second terminal of net 1 in random positions on the boundary of the rectangle. The corners of the rectangle are not considered as possible terminal positions. Additional terminals for the net are made with probability $1-\frac{1}{(t-1)}$. The number of connections for a single net on either the left or right edge of the channel is constrained to be less than or equal to one. This prevents a net from having multiple connections to the left or right edge of the channel. After net 1 is completed, the process is repeated for net 2, then 3, etc. until the fraction of positions filled is greater than or equal to c.

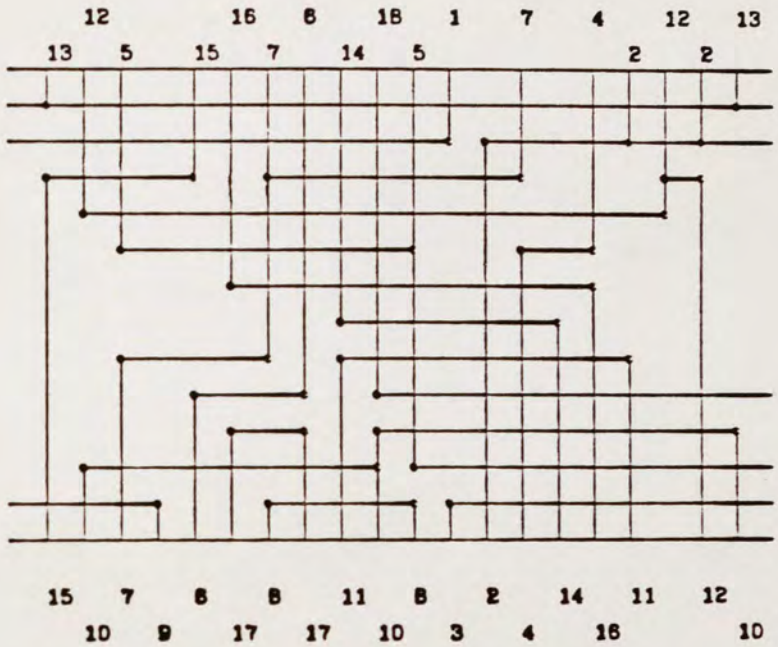


Figure 5.5. Sample channel using the Alternate algorithm

5.4. Completion Rates of Channel Routing Algorithms

Finding a single set of test cases to compare channel routing algorithms is difficult because, in practice, channel characteristics vary widely. Our approach is to generate sets of channels, each with a specific set of characteristics. If one algorithm consistently has a higher completion rate than the other algorithms over all the sets of channels generated, the strong inference is that the algorithm has better overall performance. If a particular algorithm has a higher completion rate on only a few of the sets of channels, we can isolate which channel characteristics the algorithm requires to achieve high completion rates.

Each set of test channels contains 50 sample channels. Each channel was routed using each of the six algorithms and the number

of tracks each used was recorded. The results are shown using a graph that plots the completion rate against the *effectiveness coefficient*, ρ. Given that k is the number of tracks used in the solution and D is the density, $\rho = \frac{k}{D} - 1$. This reflects the fraction of tracks over the density required to complete the routing.

In Chapter 3, the statement was made that the Dogleg and Basic Dogleg algorithm had almost identical completion rates. As an example of the format for comparing different algorithms, Figure 5.6 compares the completion rate against the effectiveness coefficients for the two algorithms. The samples were generated by the Alternate algorithm. Each channel had 30 columns, an average of 9 nets entering the right and left side of the channel, 2.5 terminals per net and 90% of the possible terminal positions were filled. The graph

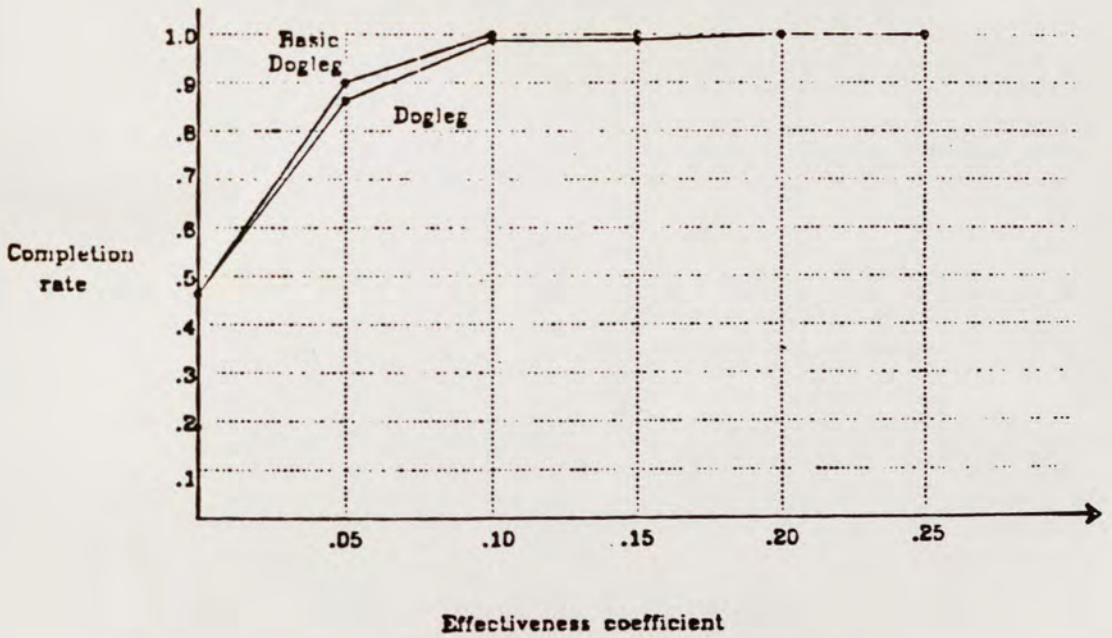


Figure 5.6. Comparison of the Dogleg and Basic Dogleg

shows that of the 50 channels, the Dogleg algorithm completed 47% in the density; 87% completed in .05 tracks over the density while 98% completed using .10 tracks over the density. The Basic Dogleg improved only slightly on these results.

The following sections compare the routing results in highly congested channels with multipoint nets, highly congested channels with two point nets, moderately congested channels with multipoint nets and channels with low aspect ratios and multipoint nets. A complete summary of all the results is provided in Appendix 1. In addition to the competition rate data presented in this section, the summaries contain breakdowns of wire and vias in the wiring, heights of vertical constraint graphs, channel utilizations, etc.

5.4.1. Highly Congested Channels with Multipoint Nets

Two sets of highly congested channels with multipoint nets were created and compared.

5.4.1.1. Channels Generated Using Rivest's Algorithm

Each channel contained 50 columns, a density of 20, 2.5 connections per net, 90% of the possible terminal positions used for terminals, and an acyclic vertical constraint graph. Figure 5.7 graphs each algorithm's completion rate against the effectiveness coefficient, ρ.

The result indicate that the LCP algorithm uses the fewest tracks over the density, followed by the Revised Dogleg, the Greedy algorithm, the Dogleg, the DOF, and the Lee algorithm.

The *effectiveness ratio* is defined as the ratio of the mean effectiveness coefficient, for each algorithm algorithm over the lowest

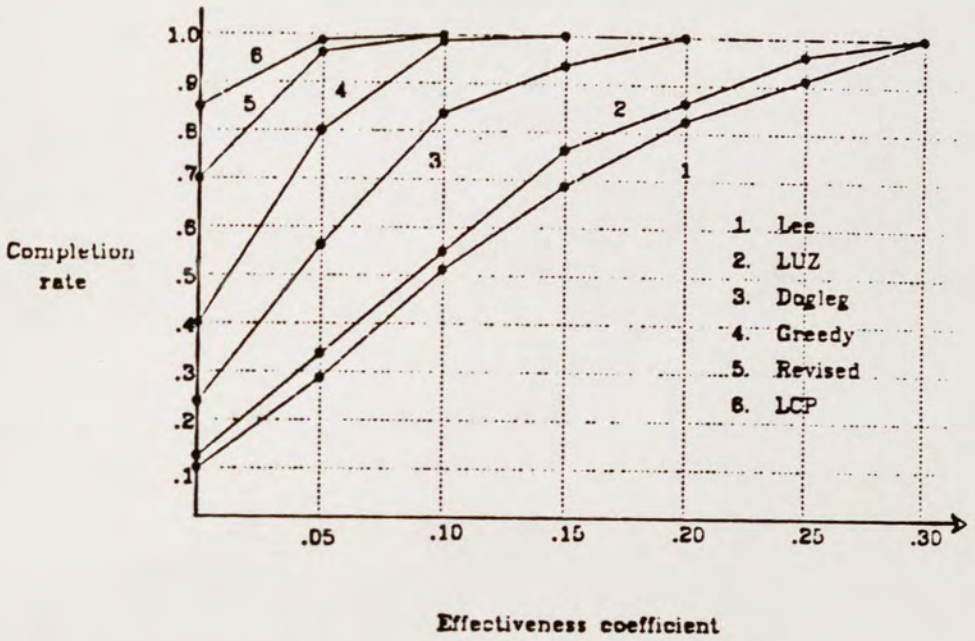


Figure 5.7. Rivest (n=50, d=20, t=2.5, c=.9, u=0, a=true)

mean effectiveness coefficient. The mean value of the effectiveness coefficients and the effectiveness ratios are given below:

TABLE 1
HIGHLY CONGESTED CHANNELS WITH MULTIPOINT NETS GENERATED USING RIVEST'S ALGORITHM

	Dogleg	Greedy	DOF	Lee	Revised	LCP
$\bar{\rho}$	.071	.04	.121	.133	.017	.009
$\frac{\bar{\rho}}{.009}$	9.889	4.44	13.444	14.778	1.88	1

This table indicates that over these test cases, the Dogleg algorithm required almost 10 times as many tracks over the density as the LCP algorithm. The DOF algorithm, the best of the two area routing algorithms, required over 13 times the number of tracks over the density.

Section 5.1 pointed out that the the Dogleg based algorithms do an excellent job of taking into consideration vertical constraints and doing line segment ordering, but they use a limited set of path shapes. It is interesting to contrast this with the Lee and DOF algorithms that do a poor job of line segment ordering, a fair job of handling vertical constraints, but have at their disposal an unlimited selection of path shapes. The above statistics indicate that complex path shapes are not as important as other factors. In fact, statistics kept for the DOF algorithm over these 50 test cases indicate less than 1.4% of all the paths used by the DOF algorithm contained 3 or more vias!

5.4.1.2. Channels generated using the Alternate Algorithm

The density mean for the channels generated by the Alternate algorithm is 19.6. The expected number of connections per net is 2.5. The fraction of positions with pins is .9, and the number of columns is 30. Figure 5.8 graphs each algorithm's completion rate against the effectiveness coefficient.

The mean value of the effectiveness coefficients and the effectiveness ratios are given in Table 2.

TABLE 2
HIGHLY CONGESTED CHANNELS WITH MULTIPOINT NETS
GENERATED USING THE ALTERNATE ALGORITHM

	Dogleg	Greedy	DOF	Lee	Revised	LCP
$\bar{\rho}$	.0377	.0221	.0918	.0864	.0093	.004
$\frac{\bar{\rho}}{.004}$	9.425	5.525	22.95	21.6	2.325	1

It is interesting to note that although the relative positions of the algorithms did not change from the statistics in the previous section, each algorithms completion rates significantly improved. The

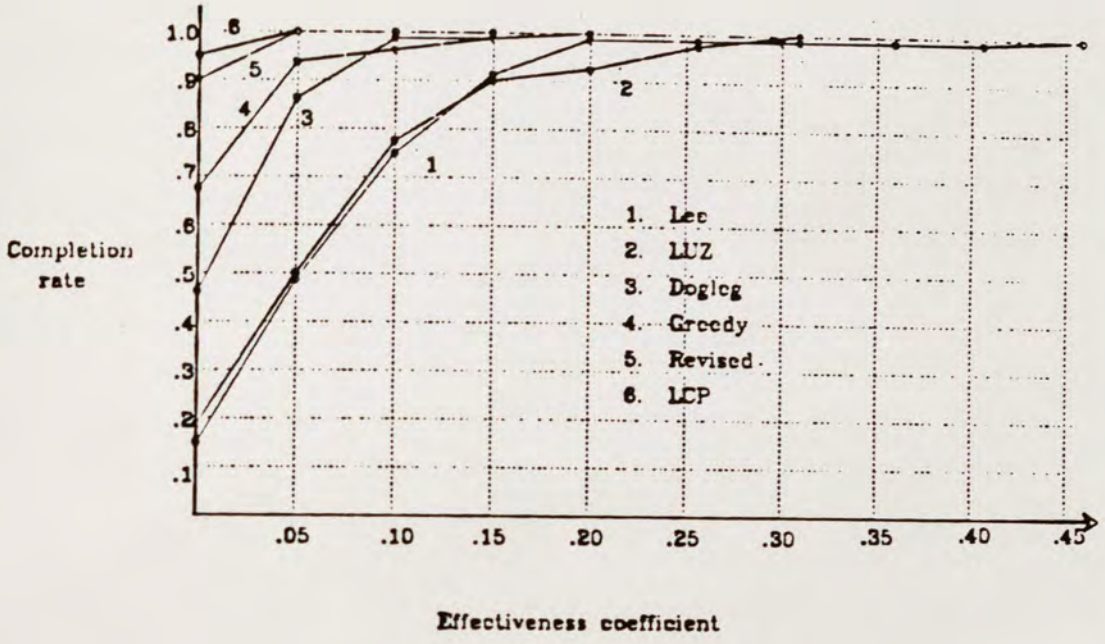


Figure 5.8. Alternate (n=30, l=10, t=2.5, c=.9, a=true)

statistics generated during the routing reveal several possible explanations. First, the number of columns in these examples was 30 rather than the 50 column examples generated by Rivest's algorithm with the density approximately the same. Second, the channels generated using the alternate algorithm tend to be hourglass shaped, leaving areas of white space in the routing region. One possible measure of this white space is the *channel utilization*, γ. γ is defined as the sum of the minimum lengths of all horizontal segments divided by the sum of the lengths of the horizontal tracks. Thus γ indicates the fraction of the total horizontal tracks that must be used to complete the wiring. The channels generated by Rivest's algorithms in section 5.4.1.1 had *gamma* =.77 with standard deviation of .12, as compared to the Alternate algorithms *gamma* =.73 and standard

deviation of .04. Furthermore, 54% of the Rivest algorithm channels had very difficult channels ($\gamma \geq .80$) as opposed to only 4% of the channels generated by the Alternate algorithm.

5.4.2. Highly Congested Channels with Two Point Nets

Zahir[Za82] examined completed custom layouts of student projects at Stanford University and found that 80% of all nets (with the exception of power, ground and clock lines) were made up of two point nets. In the following sections, Rivest's and the Alternate algorithms are again used to construct sets of sample channels consisting solely of two point nets.

5.4.2.1. Channels Generated Using Rivest's Algorithm.

The channels generated again have 50 columns, a density of 20, a fraction of possible terminal positions used of .9 and an acyclic vertical constraint graph. All nets are two point nets. Figure 5.9 graphs each algorithm's completion rate against the effectiveness coefficient.

The mean value of the effectiveness coefficients and the effectiveness ratios for each algorithm are given in Table 3.

TABLE 3
HIGHLY CONGESTED CHANNELS WITH TWO POINT NETS
GENERATED USING RIVEST'S ALGORITHM

	Dogleg	Greedy	DOF	Lee	Revised	LCP
$\bar{\rho}$	.024	.014	.023	.032	.002	.002
$\frac{\bar{\rho}}{.002}$	12.0	7.0	11.5	16.0	1	1

Although the results again indicate that the relative positions of the algorithms remain the same, a comparison of these results with

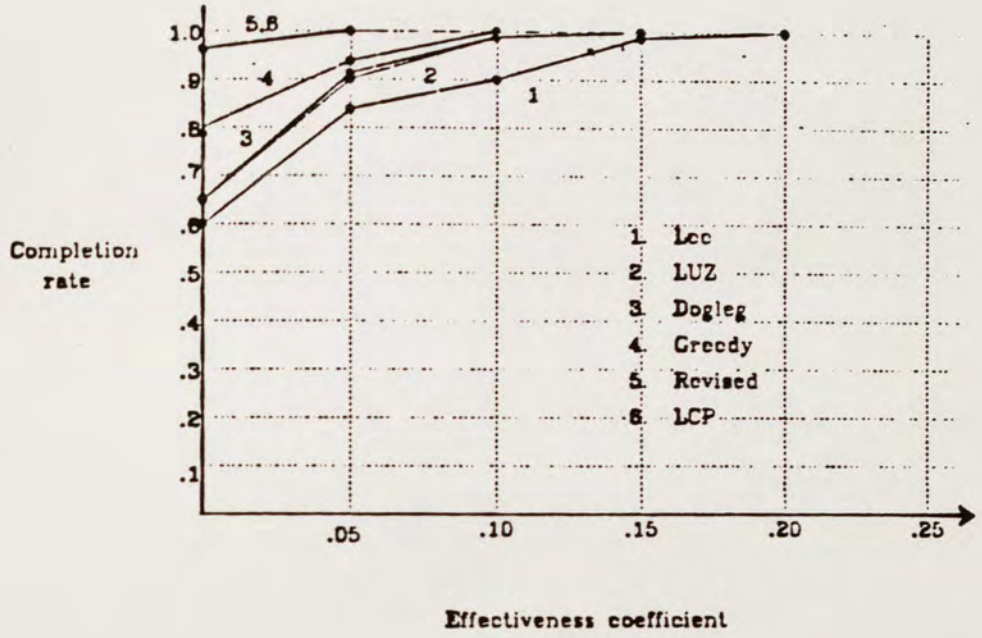


Figure 5.9. Rivest (n=50, d=20, t=2.0, c=.9, u=0, a=true)

those of section 5.4.1.1 also indicate a substantial improvement in completion rates for all the algorithms when routing channels that consist solely of two point nets. This improvement can be traced to the vertical constraint graphs of routing problems with only two point nets. As was shown in Chapter 3.3, if no vertical constraints are present, the Left edge algorithm always produces an optimal solution. Vertical constraints present obstacles to channel routing algorithms.

A t point net generally has $t-2$ interior connection points, i.e. connection points that connect two line segments of the net. A connection point in column i generates vertical constraints to all line segments that begin or end in column i and that do not belong to the same net. If a routing problem has only two point nets, a single column i can have at most two line segments that make connections

in that column. Thus the maximum number of vertical constraints created is 1. For multipoint nets column i could have up to four line segments that terminate in a single column (assuming both top and bottom connections in column i are interior nodes of multipoint nets) generating up to four vertical constraints. Figure 5.10(a) and (b) illustrate the two cases.

This increase in the number of vertical constraints was reflected in channels generated using the different values for t. For t=2.0, the mean number of vertical constraints among the test cases was 37.64; for t=2.5, the mean was 57.32.

5.4.2.2. Channels Generated Using the Alternate Algorithm

This set of channels again restricts the number of terminals per net to two. The number of columns is 30 and the fraction of terminal positions used is .9. The density mean for the channels generated is 21.4. Figure 5.11 graphs each algorithms completion rate against the effectiveness coefficient.

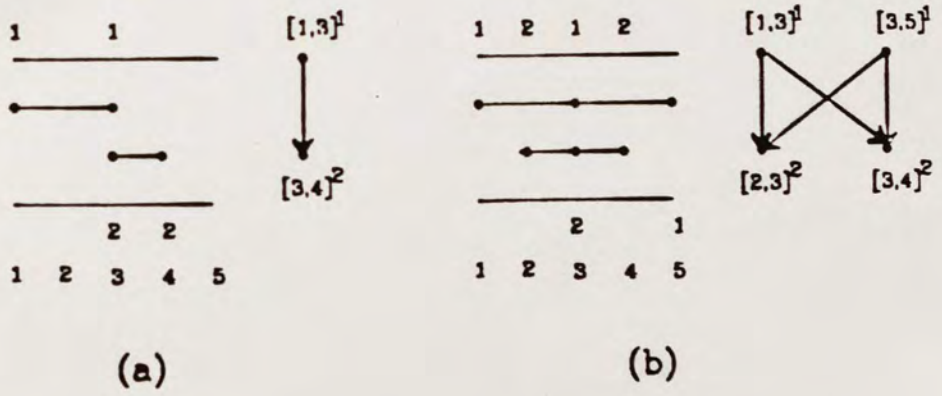


Figure 5.10. Effect of multipoint nets on vertical constraints

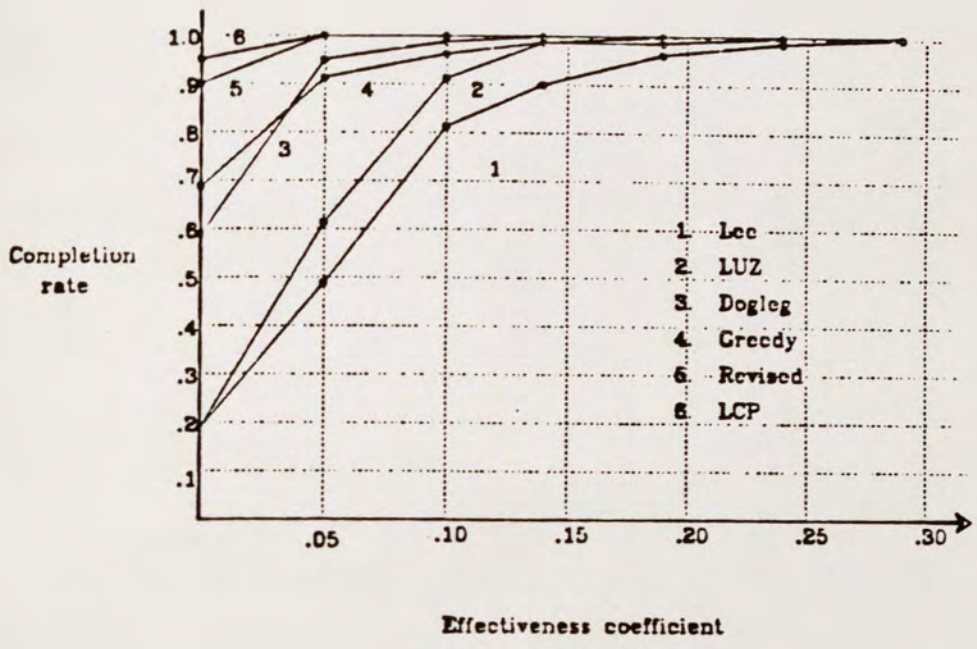


Figure 5.11. Alternate (n=30, l=10, k=2.0, c=.9, a=true)

The mean value of the effectiveness coefficients and the effectiveness ratios are given in Table 4.

TABLE 4
HIGHLY CONGESTED CHANNELS WITH TWO POINT NETS
GENERATED USING THE ALTERNATE ALGORITHM

	Dogleg	Greedy	DOF	Lee	Revised	LCP
$\bar{p}$	.0249	.0229	.0619	.0809	.005	.0027
$\frac{\bar{p}}{.0027}$	9.222	8.481	22.926	29.963	1.85	1

In these examples, the channels generated by the Alternate algorithm had higher completion rates than those generated by Rivest's algorithm. This was surprising since in section 5.4.1, the channels generated by Rivest's algorithm had higher completion rates. In that section, we noted that a possible cause of the completion rate

difference was that the Alternate algorithm was generating channels with a lower channel utilization factor, γ. The channel utilization factor was considered important because it measured the amount of white space in the channel. The statistics for the channels generated in this section show that Rivest's algorithm generated channel with a mean value for γ of .46. The Alternate algorithm generated channels with γ equal to .72. Thus, in this section the Alternate algorithm produced the more difficult channels as was reflected in the lower completion rates.

5.4.3. Moderately Congested Channels with Multipoint Nets

To generate this set of sample channels, only Rivest's algorithm is used with the fraction of possible pin positions used decreased to .6. The approximate number of terminals per net is 2.5, the number of columns is 50 and the density is 20. Figure 5.12 graphs each algorithm's completion rate against the effectiveness coefficient.

The mean value of the effectiveness coefficients and the effectiveness ratios are given is Table 5.

TABLE 5
LIGHTLY CONGESTED CHANNELS WITH MULTIPOINT NETS
GENERATED USING RIVEST'S ALGORITHM

	Dogleg	Greedy	DOF	Lee	Revised	LCP
$\bar{p}$	.06	.034	.095	.079	.005	.003
$\frac{\bar{p}}{.003}$	20.0	11.333	31.667	26.333	1.667	1

A comparison of these results with those of section 5.4.2. indicates that decreasing the value of c increases the completion rates. One reason for this improvement is decreasing c also decreases the number of vertical constraints since vertical constraints are only

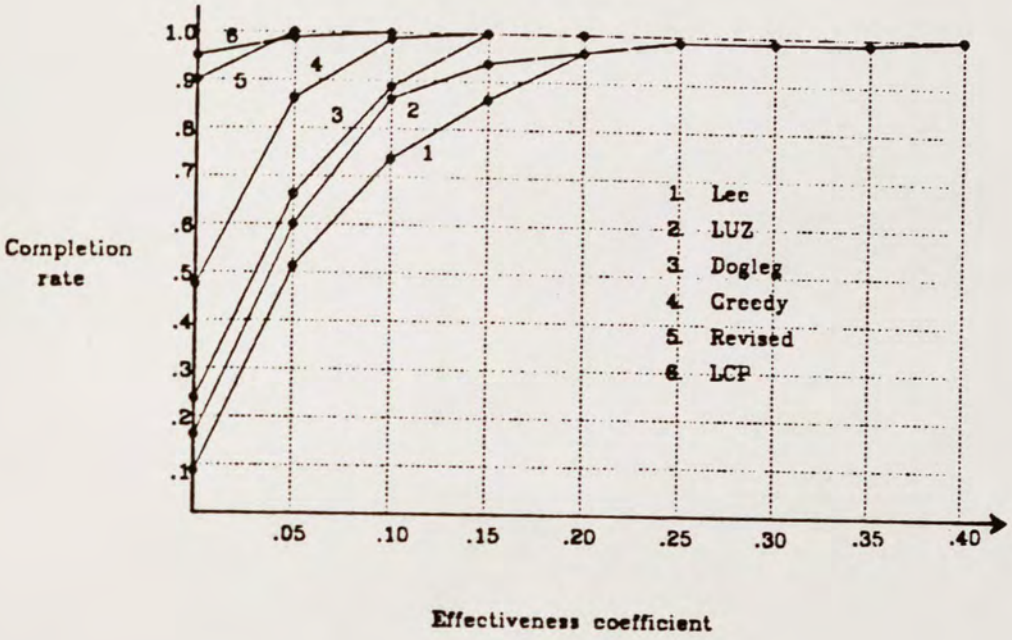


Figure 5.12. Rivest (n=50, d=20, t=2.5, c=.6, u=0, a=true)

generated when different nets connect to the top and bottom of column i. Using a value of $c = .9$, the probability of generating at least one vertical constraint is c^2 or .81. Decreasing c to .6 decreases this probability to .36.

This decrease in the number of vertical constraints can be measured by comparing the number of edges in the vertical constraint graph. In section 5.4.2, the average number of edges in the vertical constraint graph with $c = .9$ was 57.32 while in this section, with $c = .6$, the number of edges was reduced to 26.02.

5.4.4. Channels with Low Aspect Ratios and Multipoint Nets

In this example Rivest's algorithm is again used to generate the test channels but the aspect ratio (D/n) of the region is changed to

5.5. Observations on Algorithm Performance.

The purpose of this chapter was to compare each routing algorithm over a variety of channels with different characteristics and evaluate the relative performance of each. The results show that, over every set of sample channels, the relative performance of the algorithms remained the same: the best performance was recorded by the LCP algorithm, followed by the Revised algorithm, the Greedy algorithm, the Dogleg algorithm, the DOF algorithm, and finally the Lee algorithm.

The measurements also suggest that the new algorithms that evolved from the the application of Theorem 3.1 improved the number of tracks necessary to complete the routing by a factor ranging from 2.74 to 20, depending on the characteristics of the channel, with an average improvement of 10. The measurements also demonstrate a consistent area penalty when the area routing techniques of the Lee and DOF algorithms are applied to the channel routing problem. The number of tracks over the density required by the Lee and DOF algorithms ranged from 4.17 to 31.67 times the number of tracks required by the LCP algorithm with a mean of 18.32. This suggests that in channel routing problems, complex path shapes are not nearly as important as line segment alignment. The higher completion rates of the channel routing algorithms are even more impressive in light of the order of magnitude execution time penalties for area routers.

CHAPTER 6

Measurements of Algorithm Performance over Areas

Chapter 4 described the MAM and its use in evaluating the performance of an algorithm. Smith effectively used this measure to compare the completion rates of the LUZ and Lee algorithm using randomly generated square areas with two point nets. However, routing regions found in custom integrated circuits are almost always rectangular shaped with some combination of two and multipoint nets. This chapter evaluates the effectiveness of area routing algorithms in this domain.

Two important extensions were described in Chapter 4 to facilitate this comparison: the MAM was extended to multipoint nets, and the DOF algorithm was implemented that is functionally equivalent to the LUZ algorithm but is extensible to multipoint nets.

This chapter compares the Lee algorithm to the DOF algorithm over areas with a variety of aspect ratios using areas that contain both two point and multipoint nets. The results will show that the DOF algorithm has higher completion rates than the Lee algorithm over each set of sample areas. The measurements also indicate that in each set of sample areas, the completion rates fall off as the MAM exceeds .3 and that when the MAM exceeds .75, neither algorithm can successfully complete the routing. Is .75 a "barrier" intrinsic in the area routing problems generated or ineffective routing algorithms? This section defines "density" in terms of the area routing problem and establishes a relationship between the MAM and the density of the

region. This relationship defines improved upper bounds on algorithm completion rates.

6.1. Generation of Sample Areas

To compare area routing algorithms, sample areas are generated using an algorithm that is similar to the Alternate algorithm described in Chapter 5.3.2. The algorithm takes as input the following parameters:

n - area width
h - area height
t - approximate number of terminals per net. If t=2, nets are restricted to two point nets.
m - MAM for sample areas
r_0 - initial random seed.

The generation algorithm begins with a rectangular area with height $h+1$ and width $n+1$. The algorithm places the first and second terminals in random positions along the boundary of the rectangle. The corners of the rectangle are never selected as terminal positions. Additional terminals for the net are made with probability $1-\frac{1}{(t-1)}$. After net 1 is completed, the process is repeated for nets 2, then 3, etc. until the MAM of the region exceeds the input parameter m. The MAM is computed using methods detailed in Chapter 4 and provides a lower bound on the number of grid points necessary to complete the area.

Figure 6.1 illustrates a sample area generated using the parameters n=40, h=20, t=2.5 and m=.6. The wiring was produced by the DOF algorithm.

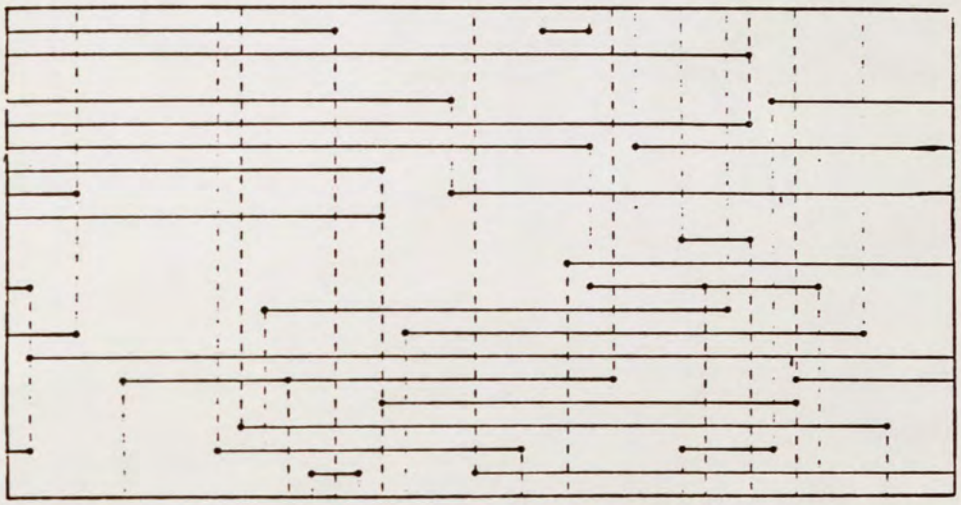


Figure 6.1. Sample area: (n=40, h=20, t=2.5, m=.5)

6.2. Comparison of the DOF and Lee Algorithms

This section graphs the completion rate of each algorithm against MAM of the sample areas. Since differing aspect ratios and multipoint nets are common in area routing problems, these statistics will emphasize areas of this form. Each data point in the graph consists of a minimum of 50 randomly generated test cases. A complete summary of these routing results is found in Appendix 2.

6.2.1. High Aspect Ratio (30x30) Regions with Two Point Nets

Smith confined the comparison of the Lee and LUZ algorithms to square routing regions with two point nets. These test cases are used

to confirm his results. In Figure 4.16, his statistics comparing the Lee and LUZ algorithms were presented. Figure 6.2 graphs the completion rates of the Lee and DOF algorithms against the MAM. As Smith noted in his statistics, the measurements indicate that the completion rates of both algorithms falls off dramatically as the MAM exceeds .5. The statistics also indicate that the DOF algorithm has higher completion rates than the Lee. These results are hardly unexpected. Smith showed the LUZ algorithm had higher completion rates than the Lee algorithm for two point nets. Since the DOF algorithm is functionally equivalent to the more complicated LUZ algorithm, we expected similar completion rates. The Lee algorithm, however, had higher completion rates than Smith found in his studies. These disparities are probably the result of variations in the implementation of the two Lee algorithms. In our implementation, a variable cost Lee was used,

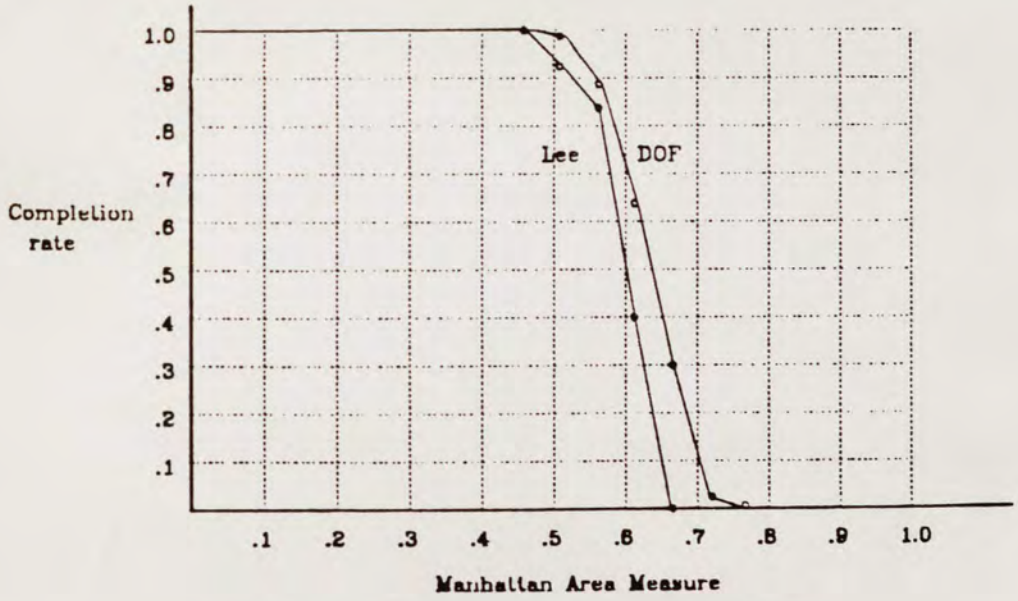


Figure 6.2. Area(n=30, h=30, t=2.0)

and very high costs were placed on via edges. This technique decreased the number of vias in the completed region and, hence, slightly increased completion rates.

6.2.2. High Aspect Ratio (30x30) with Multipoint Nets

Most area routing problems include some multipoint nets. The extensions in Chapter 4 allow the DOF algorithm to be extended to route areas with these types of nets. This section compares the effect of multipoint nets on the completion rates of the Lee and DOF algorithms using square regions. The difficulty of areas is computed using the revised definition of MAM that computes a lower bound on the number of grid points that must be used in the area divided by the total number of grid points using the lower bound on the rectilinear Steiner tree (Lemma 4.1). Both the Lee algorithm and the DOF algorithm also make use of the Steiner tree heuristic described in Chapter 4.

Figure 6.3 graphs the completion rates of these area routing problems against the MAM. The results show that extending the DOF algorithm to multipoint nets did not have an adverse affect on overall completion rates: the relative positions of the Lee and DOF algorithms remain the same.

6.2.3. Moderate Aspect Ratio (20x40) Areas with Multipoint Nets

Most areas encountered after partitioning a custom integrated circuit are rectangular shaped. The following section decreases the aspect ratio of the sample areas to $\frac{20}{40}$ or .5. The completion rates for these areas are shown in Figure 6.4. The results show that the DOF algorithm again has higher completion rates than the Lee algorithm.

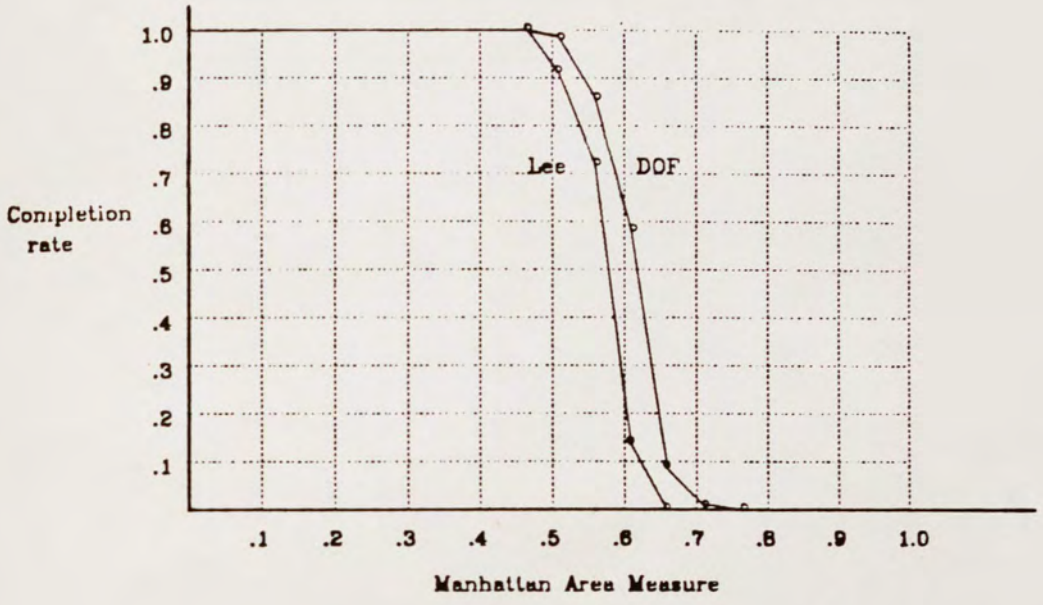


Figure 6.3. Area(n=30, h=30, t=2.5)

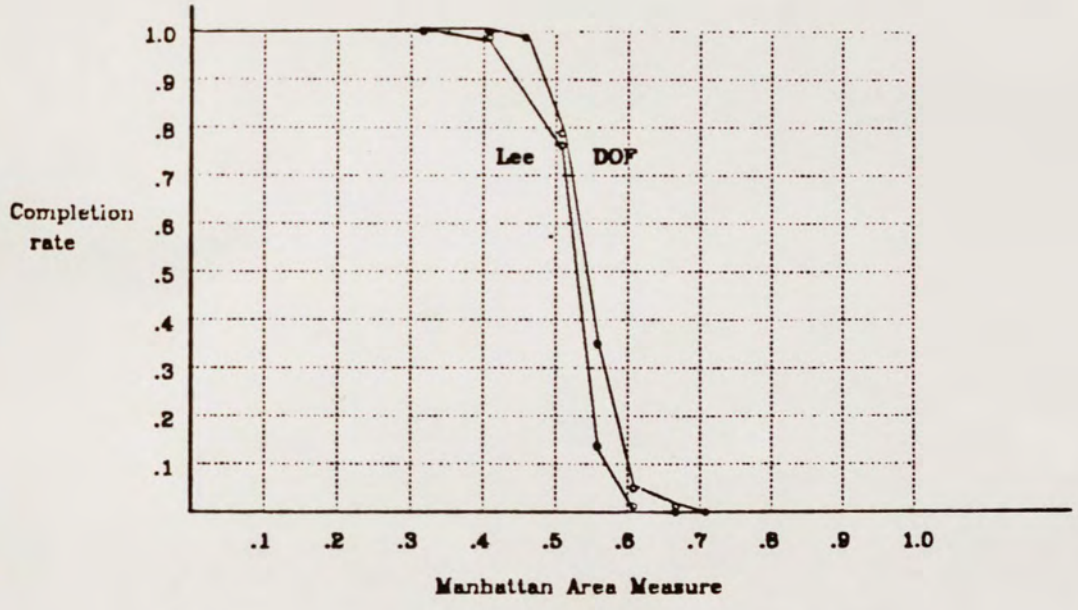


Figure 6.4. Area(n=40, h=20, t=2.5)

6.2.4. Low Aspect Areas (10x50) with multipoint nets

In these examples, the aspect ratio of the areas is decreased to $\frac{1}{5}$ or .20. The results of this comparison are shown in Figure 6.5. Two characteristics are worth noting: first, the point at which completion rates decline is less than in other aspect ratios. Second, the difference between the LUZ and DOF completion rates is considerably narrower than with previous examples.

6.3. Bounding Completion Rates

The statistics generated in the previous section indicate that each algorithm's performance diminishes rapidly when the MAM of the region reaches .3 and rarely will either algorithm route an area that has a MAM greater than .55. Is the reduction in completion rates due

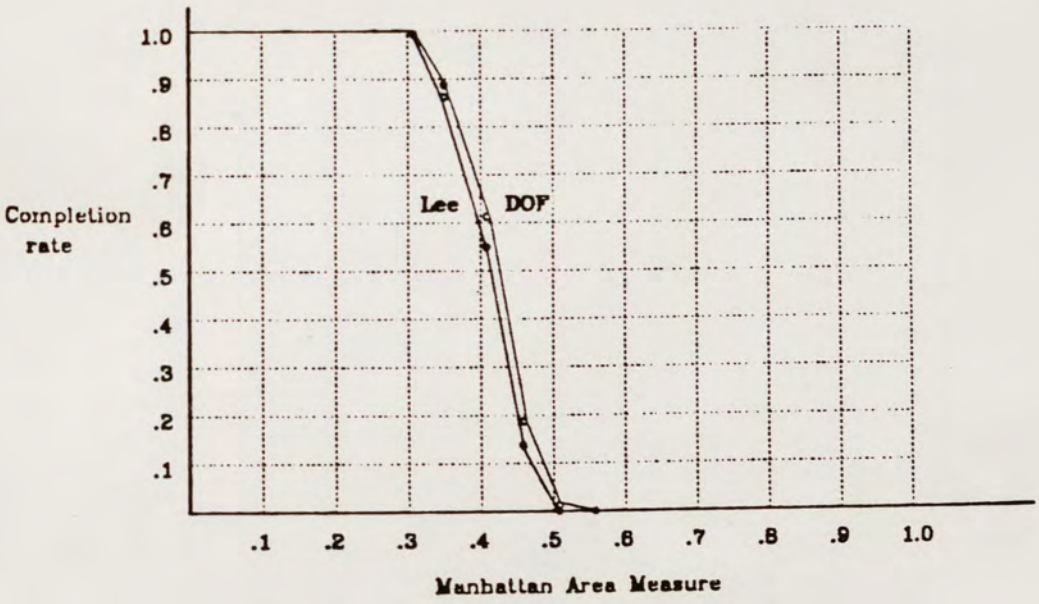


Figure 6.5. Area(n=50, l=10, k=2.5)

to poor algorithm performance, or the intrinsic difficulty of the areas attempted?

This section examines the complexity as the MAM increases and determines whether the area has reached a point were no algorithm can complete the routing. The MAM forms a simple example of this idea. Since the MAM is defined as the number of grid points that must be used to complete the wiring in an area, if the MAM exceeds 1, the area can never be routed. Unfortunately, the decrease in completion rates for the sample areas occur long before the MAM exceeds 1.0, making the MAM a poor upper bound on completion rates.

6.3.1. The Density of Area Routing Problems

A more interesting upper bound can be developed based on the concept of *density*. The definition of density is similar to the one described in Chapter 3.1. for channel routing problems. This section assumes the area is oriented such that the aspect ratio is less than or equal to one. Let the interval $p_i=[a,z]^i$ denote the set of columns crossed by net i between the leftmost terminal a and the rightmost terminal z in net i. Let the local density d_j be the number of sets where $j \in p_i$ *for* $1 \le i \le m$. Thus, as in the channel routing problem, d_j represents the number of distinct nets that cross column j. Let the area density D represent the maximum value of d_j for $1 \le j \le n$, where n is the number of columns. Since each distinct net must cross column j on a single layer, D represents a lower bound on the number of grid points that must be available in column j.

6.3.2. The Relationship Between Density and MAM

Clearly the MAM and density of a region are related. As the MAM increases, the number of nets crossing any column also increases. Let the *density coefficient* be defined as the density of the area divided by the number of horizontal tracks. For example, if the area is 10x20 and the density of the area is 8, the density coefficient is .8. Thus, 8 out of the 10 rows must be used to complete the wiring. Clearly any area with a density coefficient greater than 1 is impossible to complete, regardless of the choice of routing algorithm. Figure 6.6 graphs the density coefficient against the MAM for randomly generated areas with low aspect ratios(10x50) and multipoint nets. The center line connects the mean densities for each value of MAM. Also graphed are the mean plus or minus the standard deviation and the range of densities for each value of MAM. Notice that as the MAM exceeds .75, over 50% of the areas have density coefficients greater than 1. Thus, of the 50 test cases, less than 25 areas can be routed, regardless of the choice of routing algorithms.

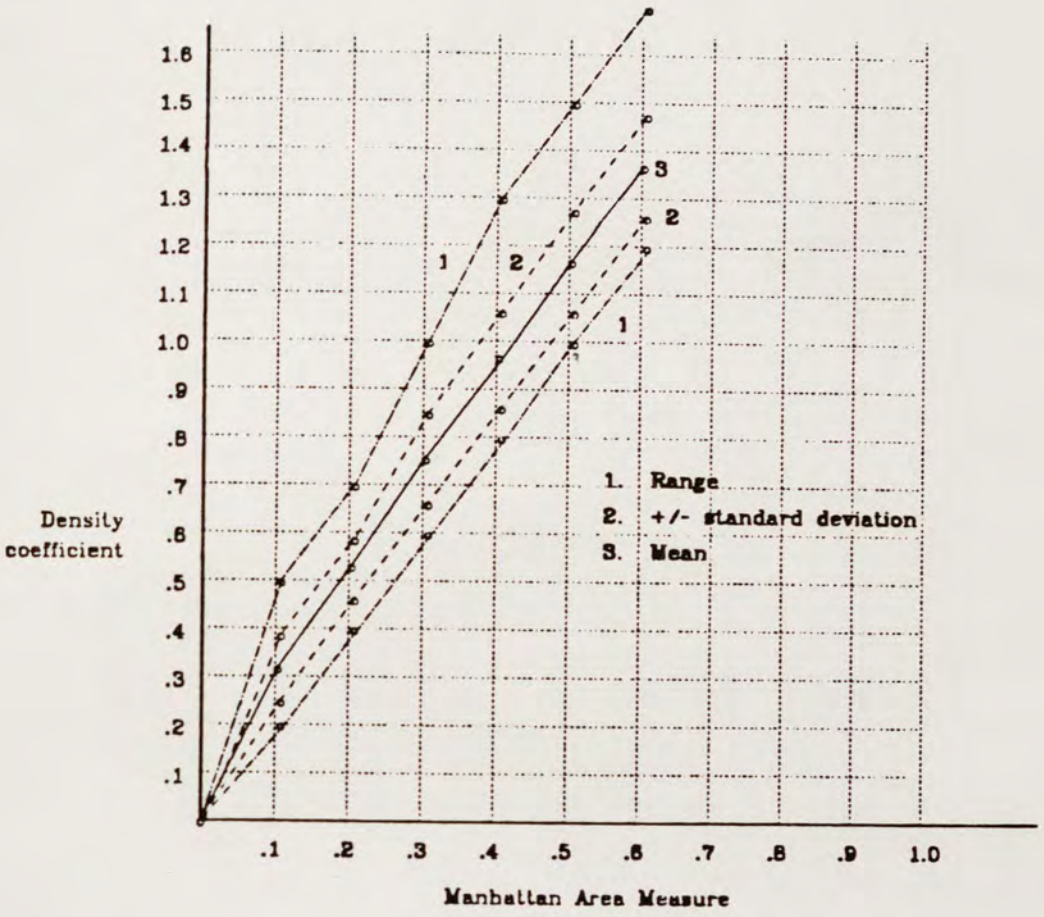


Figure 6.6. Relationship between Density and MAM

This density information can be used as an upper bound on the completion rates of area routing algorithms. Consider Figure 6.7. This graph repeats the area routing statistics from section 6.3; however, the graph has been augmented with a third line that maps the MAM to the percentage of areas where the density coefficient is less than 1. For example, when the MAM is .4, only .82 of the areas have density coefficients less than one. Thus, this line forms a upper bound on completion rates. The DOF algorithm is completing .4/.82 = .49, or 49%, of these difficult areas.

The statistics indicate that in the case of low aspect ratio areas, the decrease in completion rates is a result of the difficulty of the problems attempted rather than poor algorithm performance.

The augmented completion rate graphs for moderate and high aspect ratio areas with multipoint nets are shown in Figures 6.8 and 6.9.

These statistics indicate that the bound provided by the density is strongest in areas with a low aspect ratios and weakest in areas with and aspect ratio of 1. However, even in these cases, the density provides a much stronger bound than the MAM.

6.4. Implications of Routing Statistics

The statistics generated in this chapter have several important applications. First, they can be used to estimate the area required to complete the wiring between two adjacent modules in the custom

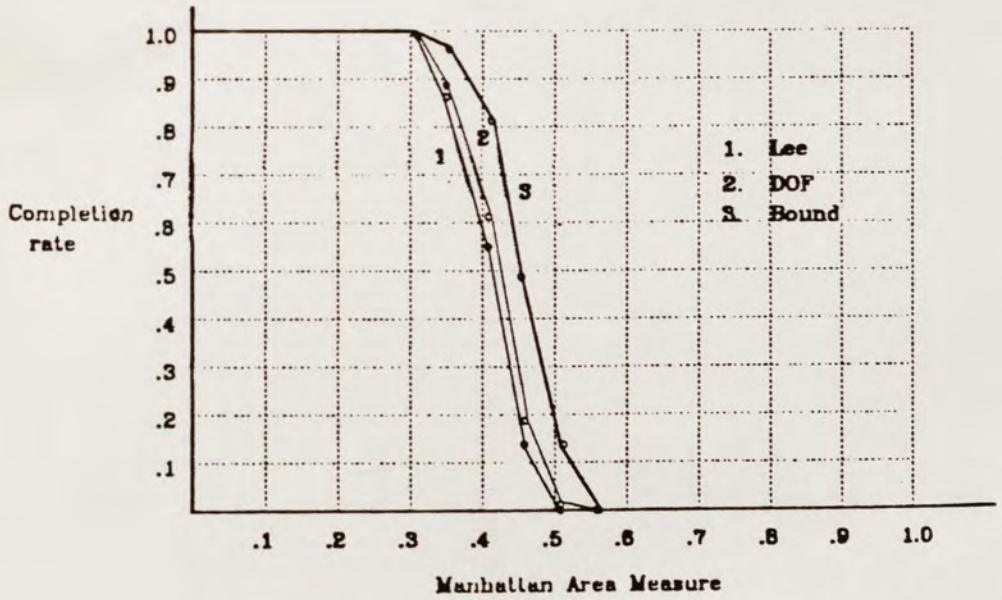


Figure 6.7. Area(d=10, n=50, t=2.5)

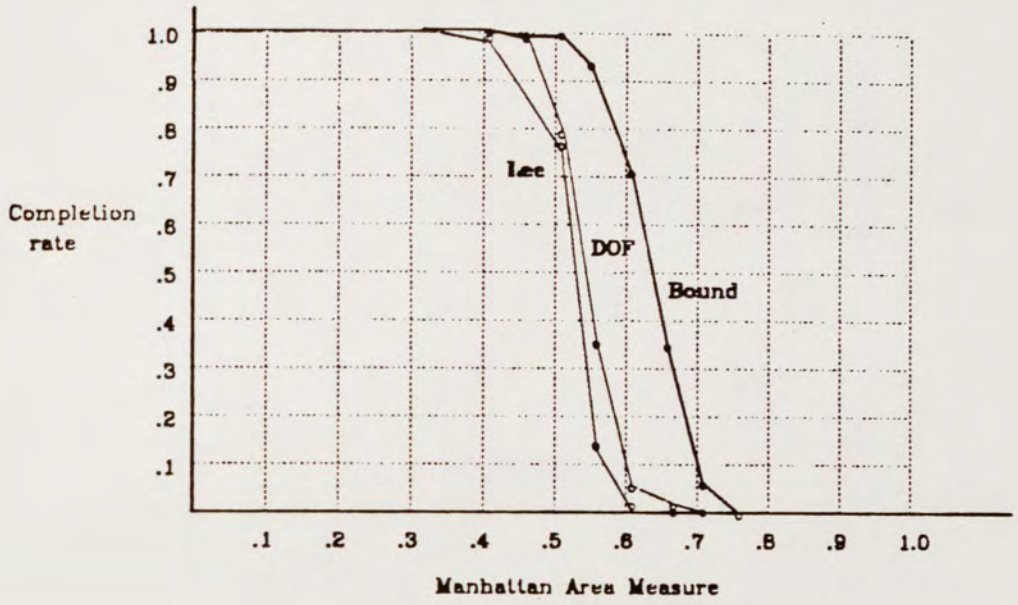


Figure 6.8. Augmented completion rate graph (n=40, h=20, t=2.5)

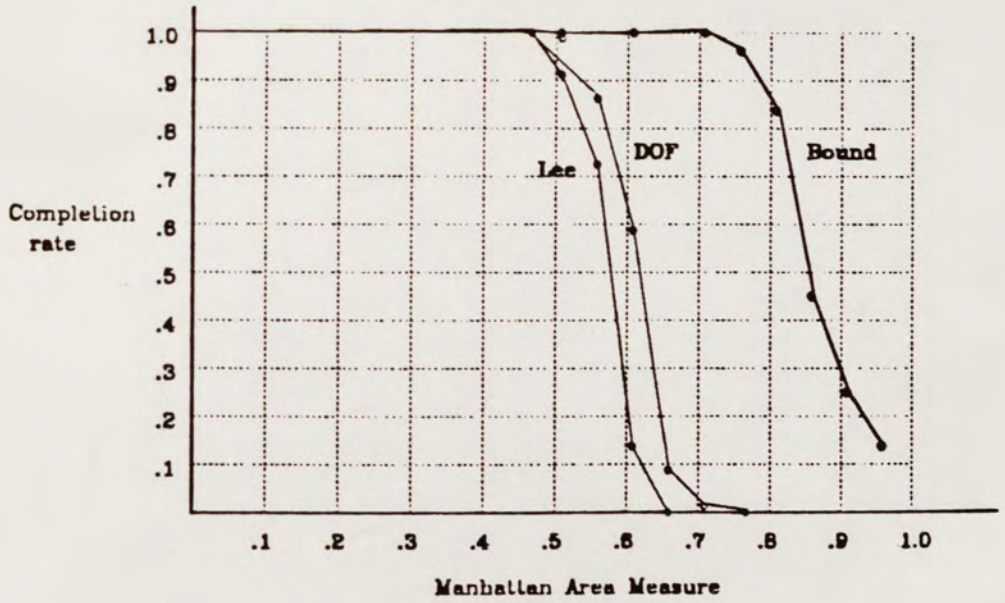


Figure 6.9. Augmented completion rate graph (n=30, h=30, t=2.5)

routing problem. For example, to have a 90% chance of completing the routing in a square region with multipoint nets, enough area should be left between modules to reduce the MAM to less than .5. If the MAM of the current module placement exceeds .75, it would be wise to alter the placement of modules to reduce the complexity of the region.

These statistics also show that the Lee and DOF algorithms perform equally well in areas up to some clearly defined threshold. For square regions, this threshold occurrs at a MAM of .5. This suggests that the added computation time required by the DOF algorithm is not necessary in such regions. In regions with MAM > .5, the improved completion rates outweigh the additional computation time.

CHAPTER 7

Conclusions and Suggestions for Further Study

This dissertation analyzes the algorithms used and performance of integrated circuit routing algorithms. This section analyzes the impact of this research on the custom routing problem. Most routing systems perform the following steps:

(1) Place the modules.

(2) Partition the routing area into rectangular routing regions. Many routing systems attempt to partition the routing region into channels. While this is laudable goal, many placements do not lend themselves to this partitioning. Consider Figure 7.1(a). The only way to partition the region into channels is as shown. This partition has two disadvantages. First, the channel graph is

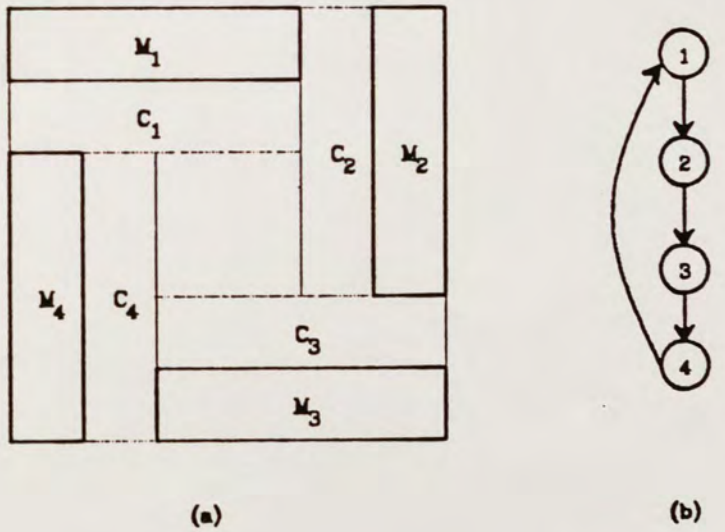


Figure 7.1. Difficult Partitioning example

cyclic, making it impossible to choose an order to route the channels. Second, it does not make use of the routing area in the center of the routing region. A more natural partition is shown in Figure 7.1(b). Also, if the modules in the layout are not rectangular, it is rarely possible to partition the routing region into solely channels. Figure 7.1(c) gives an example. This analysis indicates that natural partitions make use of both channels and areas. The Building-Block(BBL) system[ChHsKu83], developed at U. C. Berkley, is an example of a routing system that uses such a partitioning.

(3) Perform *global* routing. Global routing takes a partitioned custom circuit and determines through which regions a net will travel. Consider Figure 7.2. There are two possible paths for net A: C_1, C_2, A_1, C_3 or C_1, C_4, A_2, C_3. The choice of paths is determined by the electrical requirements of the net, combined with the congestion along the path.

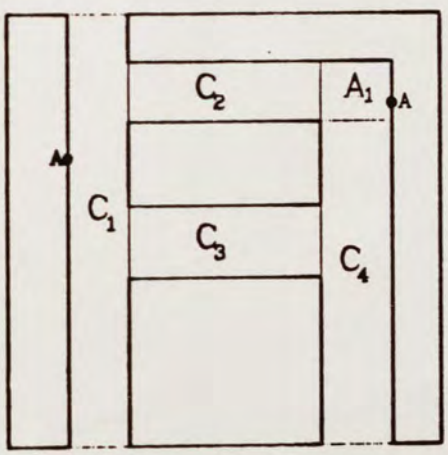


Figure 7.2. Global routing

(4) Route each region. Channels are routed using channel routing algorithms while areas are routing using area routers.

7.1. Conclusions

This dissertation contributes to each phase of the custom area routing problem.

The placement of modules depends heavily on the estimate of the amount of area required to complete the wiring. In this capacity, the MAM, in conjunction with the aspect ratio of the region, provides an excellent estimate of the chances of successfully completing the routing region.

The measure is also extremely important in the global routing phase of integrated circuit interconnection. If a low capacitance net is critical to the performance of the circuit, the shorter path is selected. Otherwise, a less-congested path that meets the electrical requirements is selected. If no path will meet these requirements, the regions must be expanded. In this context, the MAM provides an excellent estimate of the comparative difficulty of routing an additional net through the regions transversed by the two different paths. The MAM can also approximate whether or not the congestion of an area has reached a point that will make routing an additional net difficult or impossible.

Once the global routing phase has been accomplished, regions are independently routed. To complete the channels, two new and highly effective channel routing algorithms were presented: the Revised and LCP algorithms. These algorithms were shown to improve performance over the Dogleg and Greedy channel routers by a factor 2.74 to 20 with an mean of 10. The new algorithms improved the

performance of the Lee and DOF area routing algorithms by a factor of 4.17 to 31.67 times with a mean of 18.32. These statistics suggest that when "natural" channels are present in the routing region of a custom integrated circuit, area routing algorithms require significantly more area to complete the wiring than do good channel routing algorithms. This analysis further suggests that if, as in the PI routing system, each channel is made into an area by fixing the crossings, the reduced flexibility of the area routing algorithm not choosing the positions of the crossings will reduce the effectiveness of the algorithm beyond that of the previous statistics.

To complete the areas, the MAM can be used to approximate the difficulty of the routing region. For very difficult regions with high or moderate aspect ratios, the DOF algorithm should be used to complete the wiring. For regions with low aspect ratios or moderate difficulty, the results show that the faster Lee algorithm will produce the same completion rates.

7.2. Future Work

This section offers a sampling of a few of the interesting problems that remain in integrated circuit routing.

7.2.1. Applying Channel Routing Techniques to Areas

The channel routing results of Chapter 5 indicate that the Lee and DOF algorithms are not as effective as the channel routing algorithms at routing channels. Since low aspect ratio areas have few fixed pins on the left and right sides of the channel and appear as channels in all other respects, we would expect that channel routing techniques would be effective in low aspect ratio channels.

To test this hypothesis, we developed a prototype area router based on the LCP algorithm. At present, the algorithm is only applicable to areas with an acyclic vertical constraint graph (created as in channels using the pins on the top and bottom edges of the area), two pin nets and only one pin from each net entering the left or right sides of the area.

Two typical problems that make the application of channel routing techniques difficult to apply to areas are illustrated in Figure 7.3. In the first case, net one must be place in track 1. Unfortunately, net one has ancestors in the vertical constraint graph and therefore cannot be placed until the ancestors have been placed. A similar situation occurs when a net could be placed in the current track but the fixed endpin position postpones it's placement until the track corresponding to the fixed endpin. If this net has ancestors in the vertical constraint graph, they too are postponed -- often with disastrous results. Figure 7.3(b) gives a typical example.

To solve these problems, the Area LCP algorithm (ALCP) is given a set of hueristics that are applied before each track is routed. These

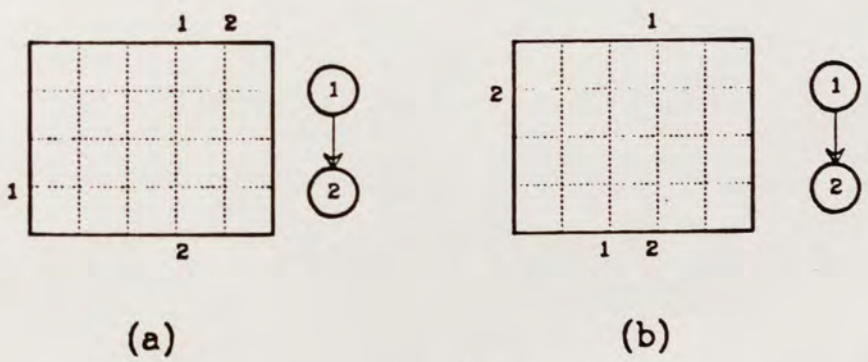


Figure 7.3. Problems with channel routing techniques

hueristics attempt to ensure that any nets with fixed endpins have no descendants in the vertical constraint graph, and thus are eligible for placement in the next track. Two techniques are used. First, the algorithm attempts to split the interval in "open" columns. Figure 7.4(a) gives an example. If this fails, the line segment is extended as shown in Figure 7.4(b). If both of these hueristics fail, the algorithm halts with failure. A similar approach is used to break line segments that have no descendants in the vertical constraint graph, but whose fixed pin position prevents the net from being placed on the current track.

The ALCP algorithm was applied to randomly generated areas that fit the previous restrictions. The results are shown in Figure 7.5. As we hypothesized, in low aspect ratio areas, the new algorithm had better results that either the Lee of the DOF algorithm -- results that are very close to the upper bound for these channels. In addition to the increase in completion rates, the ALCP algorithm was 10 times faster than the Lee algorithm and 18 times faster than the

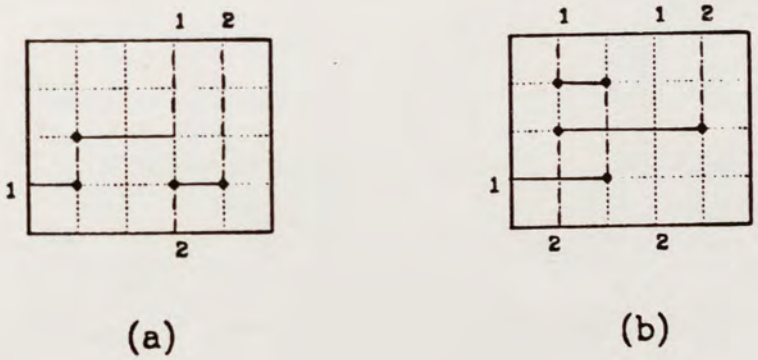


Figure 7.4. Segment breaking techniques.

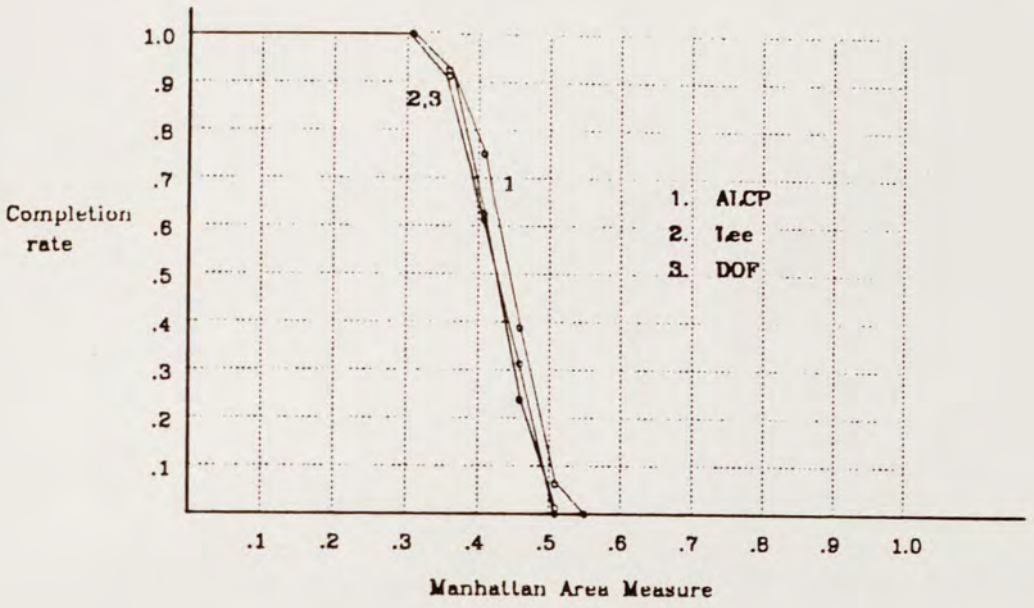


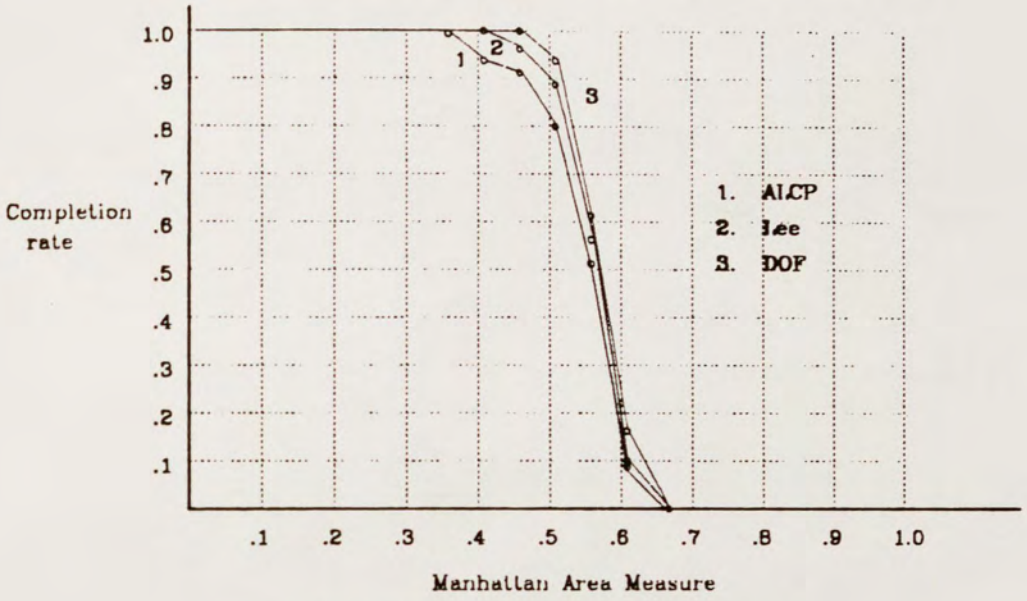
Figure 7.5. ACLP results in low aspect ration areas

DOF algorithm. A complete summary of the routing statistics is given in Appendix 3.

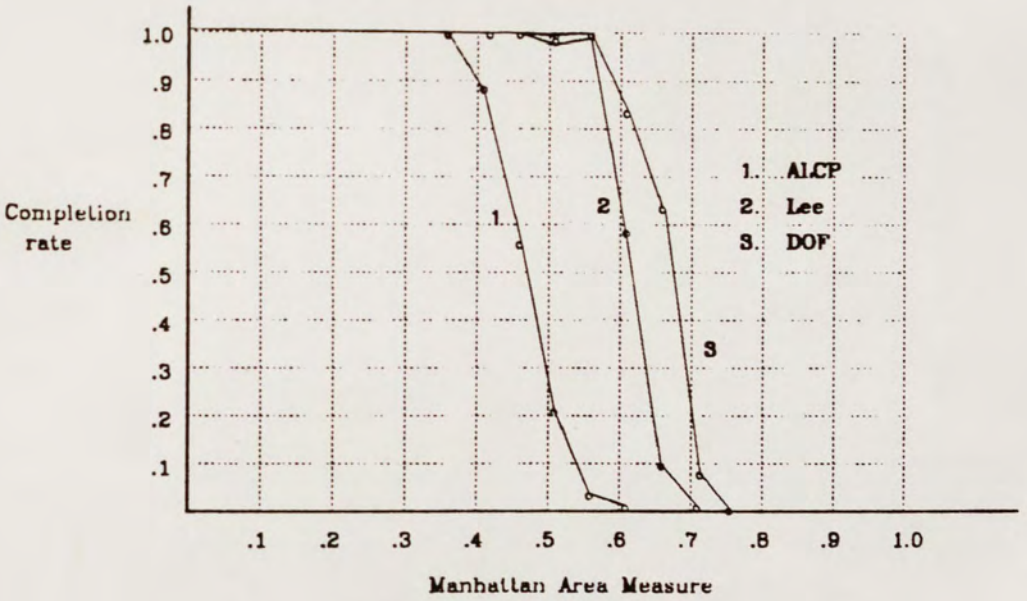
The ALCP algorithm was also applied to areas with moderate and high aspect ratios. The results are shown in Figure 7.6. As expected, as the aspect ratio increases, the number of unfixed line segments decreases and the completion rates for the LCP based algorithm also decreases.

These results indicate that area routing algorithms should use different criteria to select line segment positioning in low aspect areas.

As encouraging as the results are, the new algorithm's utility is severly restricted by the types of areas it can be applied to. In latter versions, we hope to remove the restrictions. We believe this will result in a fast, high performance area router. The algorithm will be



(a) Moderate aspect ratio



(b) High aspect ratio

Figure 7.6. ACLP results in moderate and high aspect ratio areas

most effective in low aspect ratio areas, but will also be of use in higher aspect ration areas with a low value for the MAM.

7.2.2. Soft Constraints and Preferred Positions

Many routing systems partition the routing region into channels and attempt to route each channel independently. This strategy can lead to poor routing results. Consider Figure 7.7(a). Figure 7.7(b) shows a possible routing for channel C_1 and Figure 7.7(c) shows the

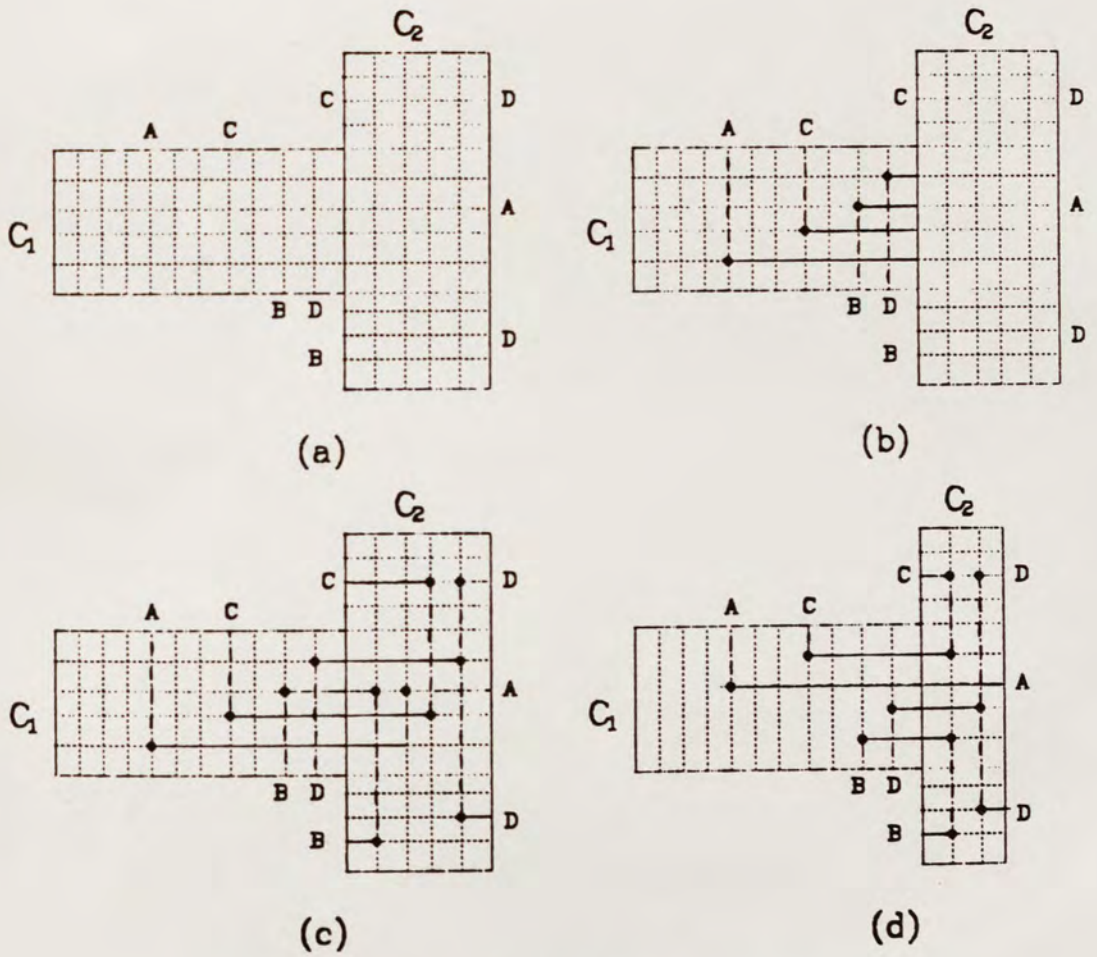


Figure 7.7. Soft constraints and preferred positions

routing for channel C_2. Had channel C_1 been routed as shown in Figure 7.7(d), channel C_2 would have been able to complete the wiring in only two tracks, rather than four tracks required in Figure 7.7(c). The problem with this routing is that the channel router did not judiciously choose the crossing points between the two regions. For this reason, the PI system fixes all the crossing between the regions using a crossing placement algorithm. This approach, while solving the problem, unnecessarily removes flexability from the routing algorithm.

Pinter[Pi81] discusses optimal solutions to the problem of crossing placements for intersecting channels; however, his approach is limited to channels with only two point nets with no vertical constraints in either channel.

An approach worthy of further investigation is based on characterizing the crossings between regions. For example, we can characterize the crossings in Figure 7.7 as follows:

(1) Preferred - a crossing has a preferred position if the placement of the endpin in this exact position places the next opposite the corresponding net in the adjacent channel. Net 2, for example, has track 4 as a preferred position.

(2) Constrained - Net 1, if placed above net 3, will decrease the density of C_2.

(3) Floating - Net 4 can be placed any free position.

The addition of these additional constraints appears to existing channel routers appears to be fairly straightforward. We believe such characterizations have all the advantages of the PI system's crossing

placement, but still maintain a high degree of algorithm flexability in choosing the exact crossing positions.

7.2.3. Elimination of Vertical Constraint Loops

If the vertical constraint graph of a channel is cyclic, neither the Dogleg, Revised Dogleg, or LCP algorithms are capable of completing the routing. In gate arrays, the placement of components can usually be rearranged to eliminate vertical constraint loops. In custom circuits, the positions of components are more difficult to change.

Vertical constraint loops are detectable by simply removing all vertices with no descendants in the vertical constraint graph, continuing the process until either no vertices remain, or no more can be removed. If no vertices remain, the graph is acyclic. Otherwise, the vertices that remain are exactly those vertices that belong to constraint loops. We believe it is possible the examine this list of vertices and, using the segment breaking techniques of section 7.2.1, remove the vertical constraints loops. These techniques might also be applicable to removing long paths in the vertical constraint graph that might impede the channel routing algorithms.

7.2.4. Metal Maximization

Metal wires have superior electrical characteristics when compared to polysilicon wires. In the routing algorithms, achieving high completion rates dictated that we limit wires in the horizontal direction to one routing layer and wires in the vertical direction to the other layer. Once the routing is completed, it would be useful to transform this wiring into one that maximizes the metal wiring layer. For example, assume that in Figure 7.8(a), the metal layer is running

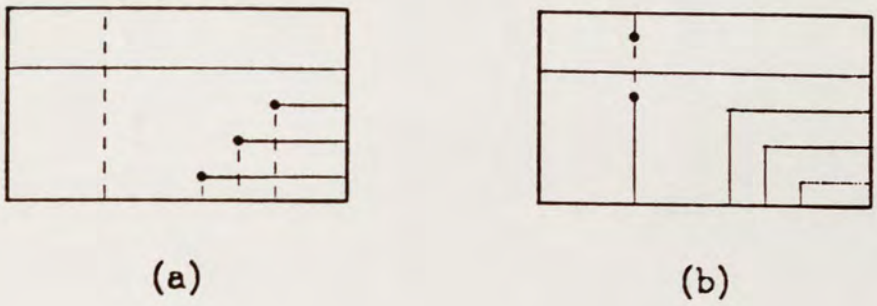


Figure 7.8. Metal layer maximization

horizontally and the polysilicon layer is running vertically. Figure 7.8(b) shows a possible transformation that increases the amount of metal wiring. An algorithm that will accomplish transformations of this type would be a valuable addition to any custom integrated circuit layout system.

7.2.5. 45° Wiring Model

In hand wired custom layouts, it is common practice to use 45° wires. Wires of this type are especially appropriate when routing groups of parallel wires. Consider Figure 7.9(a). Clearly manhattan wires would require much more area. 45° wires also have several uses in the context of channel routing. In Figure 7.10(b), the 45° wire reduces the density of the channel from 2 to 1 and eliminates the vertical constraint. Exploring algorithms that can exploit diagonal wires could easily result in a set of algorithms with vastly improved performance over traditional manhattan approaches.

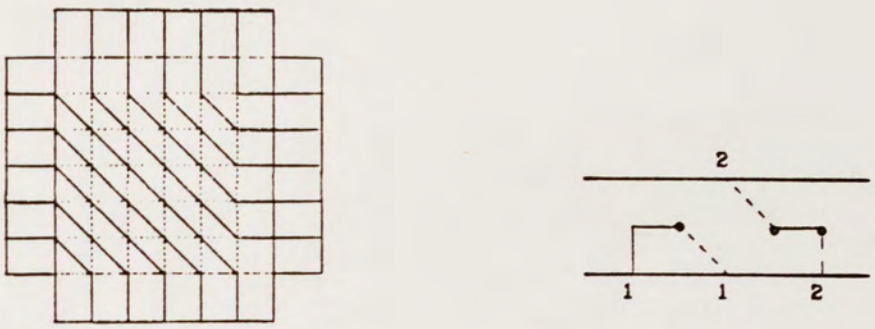


Figure 7.9. 45° wire example

7.2.6. Other Topics

The previous examples are only a few of the interesting topics that remain to be solved in custom integrated circuit routing. Other interesting topics include non-gridded routing regions, tighter bounds on area routing problems, power and ground routing, and integration of the algorithms and philosophies presented into a design environment.

APPENDIX A

Channel Routing Statistics

This appendix contains the raw data for the channel routing statistics. The following statistics have been compliled:

Density:	The density of the channel.
Actual	The mean number of tracks each algorithm used to complete the channels
Eff. Coefficient:	$\frac{Actual - Density}{Density} - 1$
Polysilicon:	The mean length of polysilicon wire.
Metal:	The mean length of metal wire.
Vias:	The mean number of vias.
Range-low:	The mean lower bound on the number of edges in the shortest path in the vertical constraint graph.
Range-high:	The mean lower bound on the number of edges in the shortest path in the vertical constraint graph.
Opt. Utilization:	The mean number of grid points necessary to complete the metal layer.
Utilization:	The mean number of grid points used to complete in the metal layer.
Height:	The number of edges in the average path in the vertical constraint graph. This average is computed by labeling each node with the maximum distance to a root node. The labels on the leaf nodes are then summed and divided by the number of leaf nodes.
Edges:	The mean number of edges in the vertical constraint graph.
Nodes:	The mean number of nodes in the vertical constraint graph.
Vias:	The mean percentage of paths that were routed using 0, 1, 2, etc. vias.

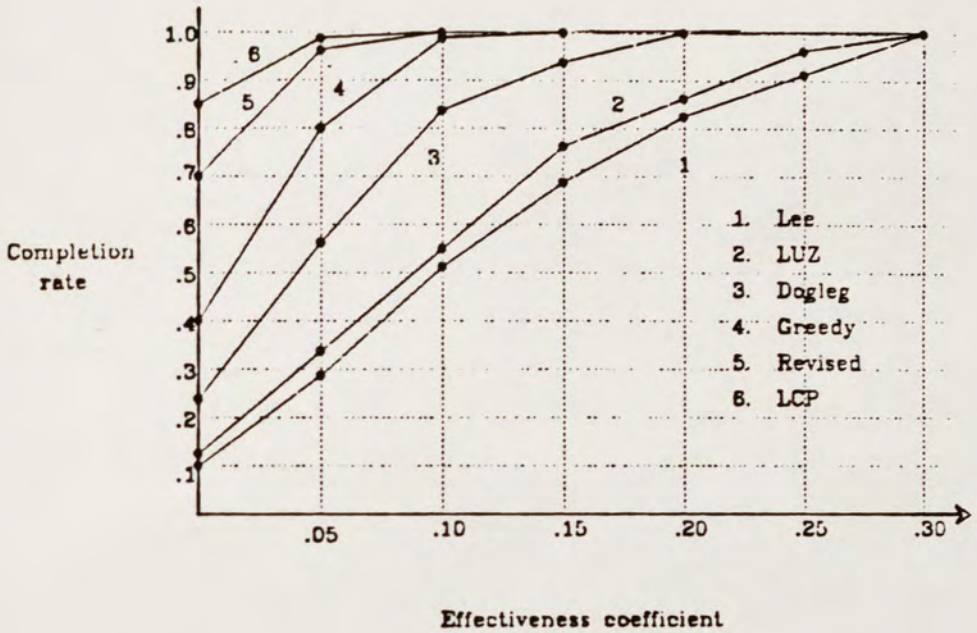


TABLE 7
HIGHLY CONGESTED CHANNELS WITH MULTIPOINT NETS
GENERATED USING RIVEST'S ALGORITHM

	LCP	Revised	Greedy	Dogleg	DOF	Lee
Density	20	20	20	20	20	20
Actual	20.18	20.34	20.82	21.42	22.42	22.6
Eff. Coef.	.009	.017	.041	.071	.1210	.133
Poly.	721.06	713.32	636.9	734.9	612.74	622.86
Metal	789.12	789.12	793.62	789.12	821.74	828.44
Vias	95.46	96.3	109.06	97.92	92.16	97.72
Range-low	.04	.04		.04		
Range-high	3.0	3.0		3.0		
Opt. Util.	.7736	.7736		.7736		
Util.	.7663	.7598		.7209		
Height	1.19	1.19		1.19		
Edges	57.32	57.32		57.32		
Nodes	70.36	70.36		70.36		
0 Via paths					.0379	
1 Via paths					.6023	
2 Via paths					.3496	
3 Via paths					.0054	
4 Via paths					.0070	
5+ Via paths					.0009	

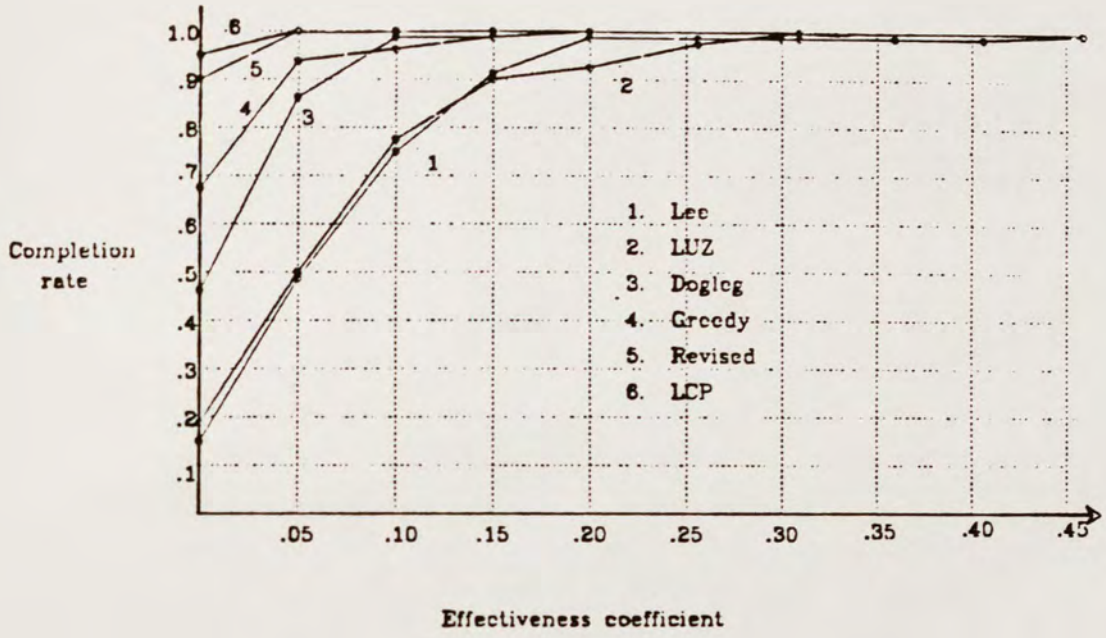


TABLE 8
HIGHLY CONGESTED CHANNELS WITH MULTIPOINT NETS
GENERATED USING THE ALTERNATE ALGORITHM

	LCP	Revised	Greedy	Dogleg	DOF	Lee
Density	19.6	19.6	19.6	19.6	19.6	19.6
Actual	19.68	19.78	20.0	20.25	21.34	21.24
Eff. Coef.	.004	.0093	.0221	.0341	.0918	.0864
Poly.	430.36	437.12	402.68	501.48	362.62	365.4
Metal	446.04	446.04	448.38	446.04	486.66	459.34
Vias	58.14	60.18	70.54	59.88	55.78	60.02
Range-low	.02	.02		.02		
Range-high	2.92	2.92		2.92		
Opt. Util.	.7366	.7366		.7366		
Util.	.7338	.7299		.7128		
Height	1.249	1.249		1.249		
Edges	37.12	37.12		37.12		
Nodes	41.92	41.92		41.92		
0 Via paths					.0589	
1 Via paths					.5283	
2 Via paths					.4010	
3 Via paths					.0009	
4 Via paths					.000	
5+ Via paths					.000	

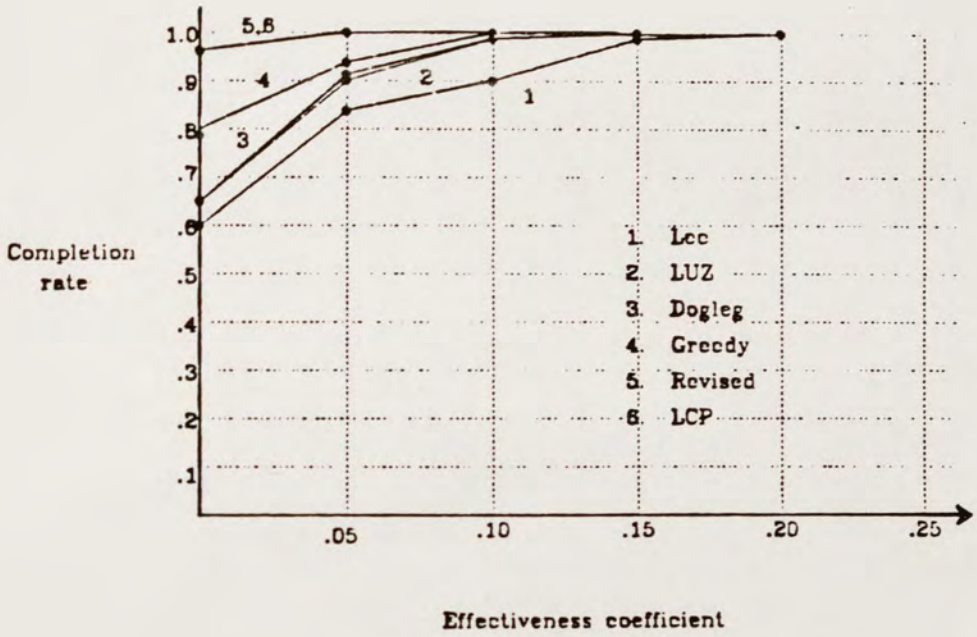


TABLE 9
HIGHLY CONGESTED CHANNELS WITH TWO POINT NETS
GENERATED USING RIVEST'S ALGORITHM

	LCP	Revised	Greedy	Dogleg	DOF	Lee
Density	20	20	20	20	20	20
Actual	20.04	20.04	20.28	20.48	20.46	20.64
Eff. Coef.	.002	.002	.014	.0240	.230	.032
Poly.	628.9	609.74	529.24	596.78	515.18	518.38
Metal	477.28	477.28	479.88	477.28	477.48	485.48
Vias	84.46	84.46	100.5	84.46	86.18	90.34
Range-low	.04	.04		.04		
Range-high	2.94	2.94		2.94		
Opt. Util.	.4679	.4679		.4679		
Util.	.4666	.4666		.4541		
Height	1.147	1.147		1.147		
Edges	37.64	37.64		37.64		
Nodes	54.34	54.34		54.34		
0 Via paths					.050	
1 Via paths					.411	
2 Via paths					.5285	
3 Via paths					.005	
4 Via paths					.000	
5+ Via paths					.000	

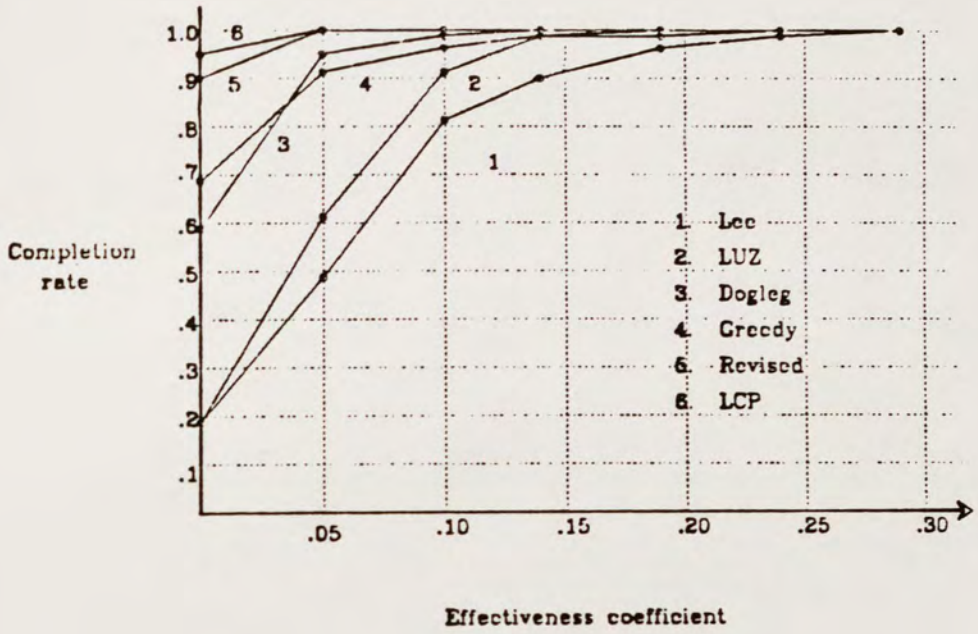


TABLE 10
HIGHLY CONGESTED CHANNELS WITH TWO POINT NETS
GENERATED USING THE ALTERNATE ALGORITHM

	LCP	Revised	Greedy	Dogleg	DOF	Lee
Density	21.04	21.04	21.04	21.04	21.04	21.04
Actual	21.10	21.14	21.48	21.54	22.32	22.7
Eff. Coef.	.0027	.005	.0229	.0249	.0619	.0809
Poly.	434.1	438.38	397.84	444.12	370.86	360.76
Metal	468.88	468.88	471.52	468.88	469.2	477.8
Vias	52.12	52.12	65.08	52.12	54.36	57.6
Range-low	0	0		0		
Range-high	2.7	2.7		2.7		
Opt. Util.	.7213	.7213		.7213		
Util.	.7194	.7178		.7041		
Height	1.095	1.095		1.095		
Edges	22.96	22.96		22.96		
Nodes	34.64	34.64		34.64		
0 Via paths					.0549	
1 Via paths					.4011	
2 Via paths					.5183	
3 Via paths					.0000	
4 Via paths					.0206	
5+ Via paths					.0000	

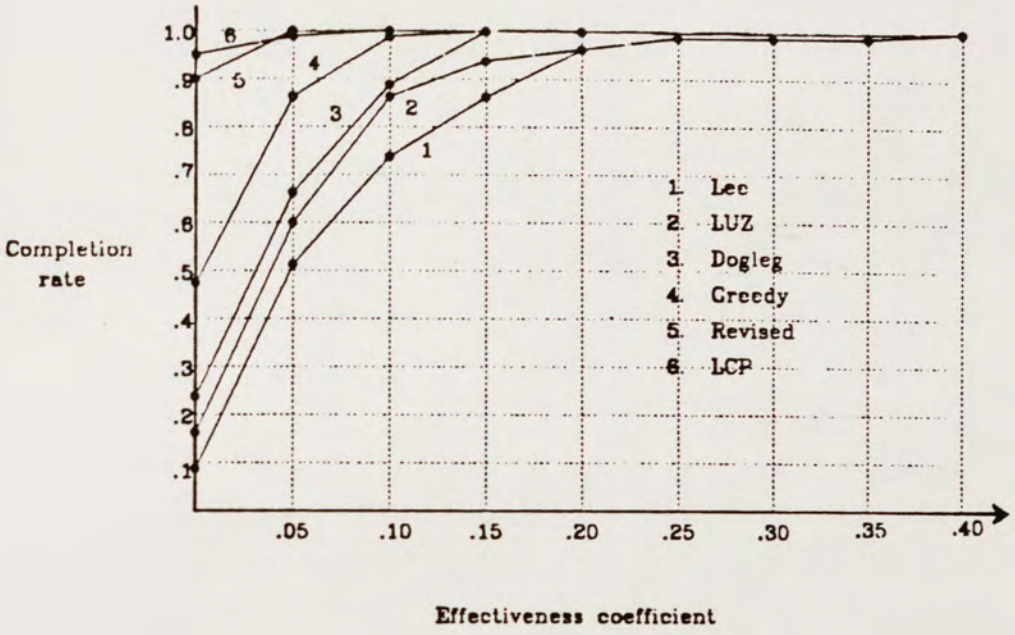


TABLE 11
LIGHTLY CONGESTED CHANNELS WITH MULTIPOINT NETS
GENERATED USING RIVEST'S ALGORITHM

	LCP	Revised	Greedy	Dogleg	DOF	Lee
Density	20	20	20	20	20	20
Actual	20.06	20.1	20.68	21.22	21.9	21.58
Eff. Coef.	.003	.005	.034	.061	.095	.079
Poly.	544.58	542.24	442.22	572.22	416.0	419.96
Metal	856.24	856.24	857.24	856.24	884.04	876.7
Vias	61.26	63.62	73.60	65.28	62.52	63.92
Range-low	0	0		0		
Range-high	2.08	2.08		2.08		
Opt. Util.	.8395	.8395		.8395		
Util.	.8370	.8353		.7906		
Height	.5696	.5696		.5696		
Edges	26.02	26.02		26.02		
Nodes	53.98	53.98		53.98		
0 Via paths					.0807	
1 Via paths					.7104	
2 Via paths					.1923	
3 Via paths					.003	
4 Via paths					.00	
5+ Via paths					.00	

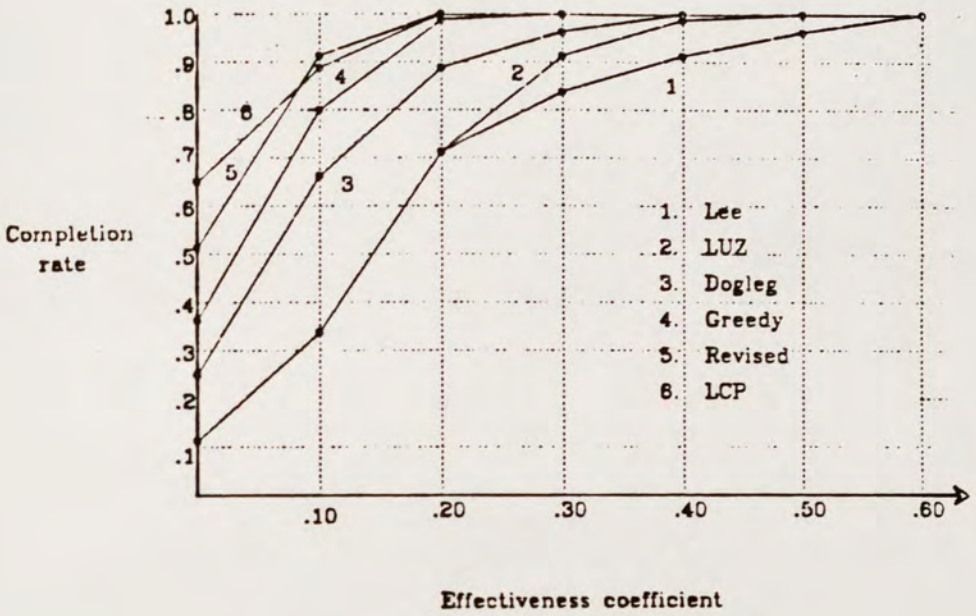


TABLE 12
HIGHLY CONGESTED, LOW ASPECT RATIO
CHANNELS WITH MULTIPOINT NETS
GENERATED USING RIVEST'S ALGORITHM

	LCP	Revised	Greedy	Dogleg	DOF	Lee
Density	10	10	10	10	10	10
Actual	10.46	10.56	10.86	11.26	11.72	12.10
Eff. Coef.	.046	.056	.086	.126	.192	.21
Poly.	361.38	362.98	351.22	373.0	344.8	355.24
Metal	318.18	318.18	321.3	318.18	366.5	339.48
Vias	92.16	93.7	107.16	94.74	89.5	94.16
Range-low	.04	.04		.04		
Range-high	3.14	3.14		3.14		
Opt. Util.	.6239	.6239		.6239		
Util.	.5937	.5881		.5515		
Height	1.344	1.344		1.344		
Edges	56.02	56.02		56.02		
Nodes	60.20	60.20		60.20		
0 Via paths					.0386	
1 Via paths					.3643	
2 Via paths					.5890	
3 Via paths					.0022	
4 Via paths					.0068	
5+ Via paths					.00	

APPENDIX B

Area Routing Statistics

This appendix contains the raw data for the area routing statistics. The following statistics have been compliled:

Completion rate:	The percentage of areas that the algorithm completed for a given MAM value
Polysilicon:	The mean length of polysilicon wire
Metal:	The mean length of metal wire
Vias:	The mean number of vias
DOF Vias	The mean percentage of paths that were routed using 0,1,2, etc. vias

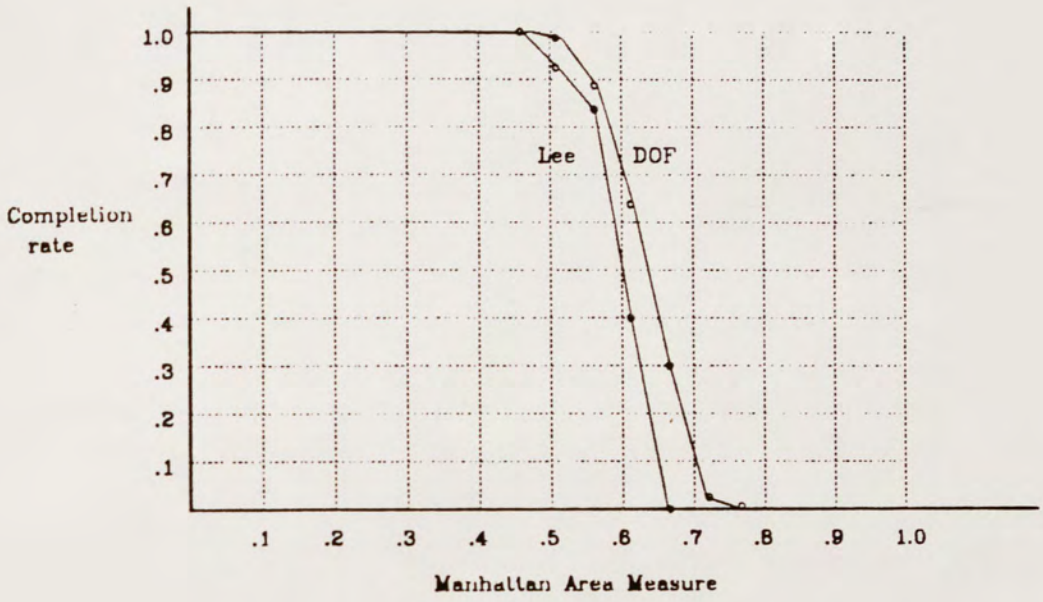


TABLE 13
HIGH ASPECT RATIO AREAS WITH TWO POINT NETS:
DEGREE OF FREEDOM ALGORITHM

	MAM					
	.45	.50	.55	.60	.65	.70
Comp. Rate	1.0	.98	.88	.64	.30	.02
Polysilicon	428.8	476.24	524.2	577.7	635.	693.
Metal	424.8	469.7	525.0	569.47	607.	657.
Vias	48.6	54.61	61.2	67.19	73.73	79
0 via path	.0077	.0077	.008	.010	.001	0.0
1 via path	.379	.365	.348	.350	.356	.523
2 via path	.482	.478	.480	.457	.413	.273
3 via path	.124	.142	.152	.162	.176	.159
4 via path	.009	.007	.002	.017	.030	.023
5+ via path	.000	.001	.000	.004	.007	.000

TABLE 14
HIGH ASPECT RATIO AREAS WITH TWO POINT NETS:
LEE ALGORITHM

	MAM				
	.45	.50	.55	.60	.65
Comp. Rate	1.0	.94	.84	.40	0.0
Polysilicon	437.0	487.4	545.8	582.5	0.0
Metal	431.0	477.	539.34	597.45	0.0
Vias	52.84	59.0	69.19	76.3	0.0

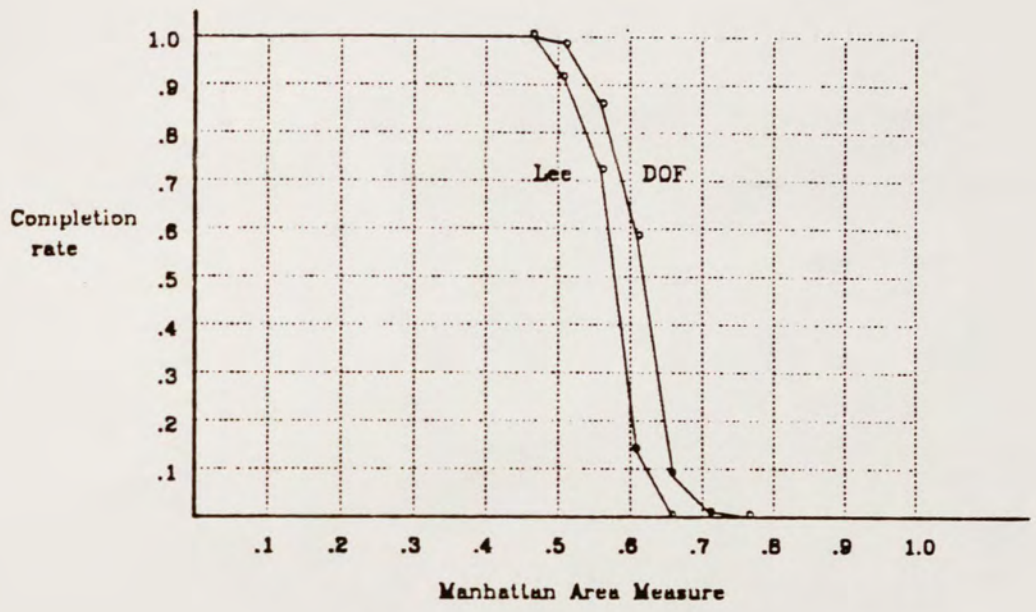


TABLE 15
HIGH ASPECT RATIO AREAS WITH MULTIPOINT NETS:
DEGREE OF FREEDOM ALGORITHM

	MAM					
	.45	.50	.55	.60	.65	.70
Comp. Rate	1.0	.98	.86	.58	.08	.02
Polysilicon	456.88	518.43	564.10	619.41	655	700
Metal	457.8	516.33	563.14	615.03	663	658
Vias	53.82	62.16	68	76.44	784	79
0 via path	.0109	.011	.010	.008	.01	0.02
1 via path	.524	.493	.477	.455	.449	.45
2 via path	.398	.408	.407	.412	.370	.396
3 via path	.066	.082	.098	.101	.133	.104
4 via path	.002	.0035	.005	.013	.026	.021
5+ via path	0.0	.0014	.001	.009	.016	0.0

TABLE 16
HIGH ASPECT RATIO AREAS WITH MULTIPOINT NETS:
LEE ALGORITHM

	MAM					
	.45	.50	.55	.60	.65	70
Comp. Rate	1.0	.94	.74	.16	.02	0.0
Polysilicon	462.86	524.89	573.86	621.62	634	0.0
Metal	463.16	522.19	581.16	607.5	596	0.0
Vias	57.2	67.02	75.8	81.37	83	0.0

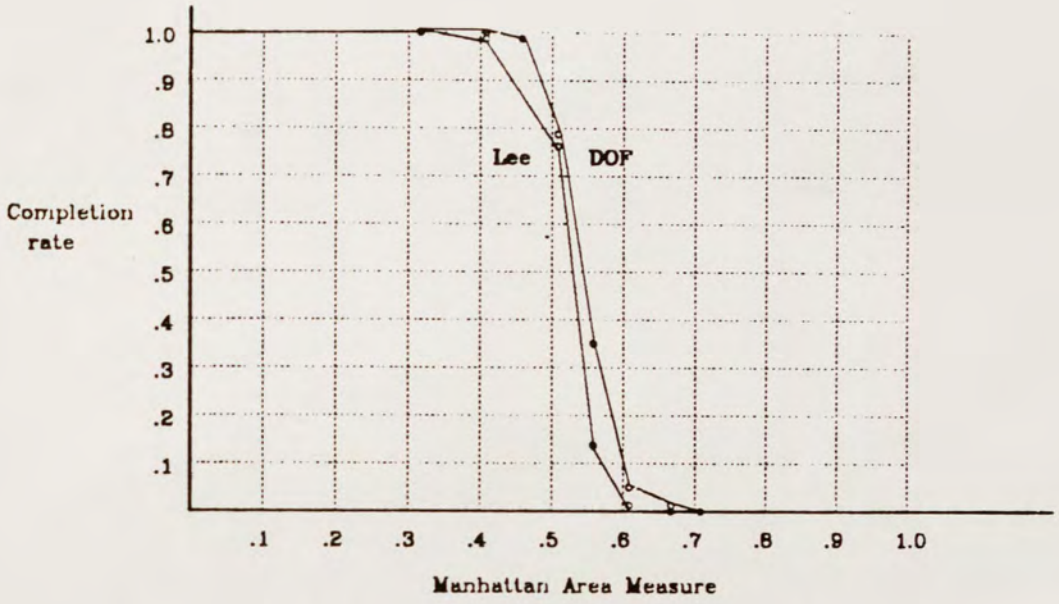


TABLE 17
MODERATE ASPECT RATIO AREAS WITH MULTIPOINT NETS:
DEGREE OF FREEDOM ALGORITHM

	MAM					
	.40	.45	.50	.55	.60	.65
Comp. Rate	1.0	.98	.78	.34	.04	.02
Polysilicon	297.32	342.29	379.77	421.35	465.5	480.0
Metal	427.54	481.86	528.12	566.65	580.5	620.0
Vias	43.96	50.39	57.28	63.29	65	68
0 via path	.0086	.009	.011	.01	.012	0.0
1 via path	.525	.500	.478	.484	.434	.42
2 via path	.419	.426	.426	.340	.410	.474
3 via path	.046	.057	.065	.079	.115	.053
4 via path	.002	.008	.015	.020	.029	.027
5+ via path	.000	.0011	.0061	.006	0.0	.026

TABLE 18
MODERATE ASPECT RATIO AREAS WITH MULTIPOINT NETS:
LEE ALGORITHM

	MAM					
	.40	.45	.50	.55	.60	.65
Comp. Rate	.98	.98	.76	.14	.02	0.0
Polysilicon	302.41	349.41	396.79	439.43	517	0.0
Metal	425.65	483.16	529.18	574.29	553	0.0
Vias	45.31	53.75	62.94	70.85	75	0.0

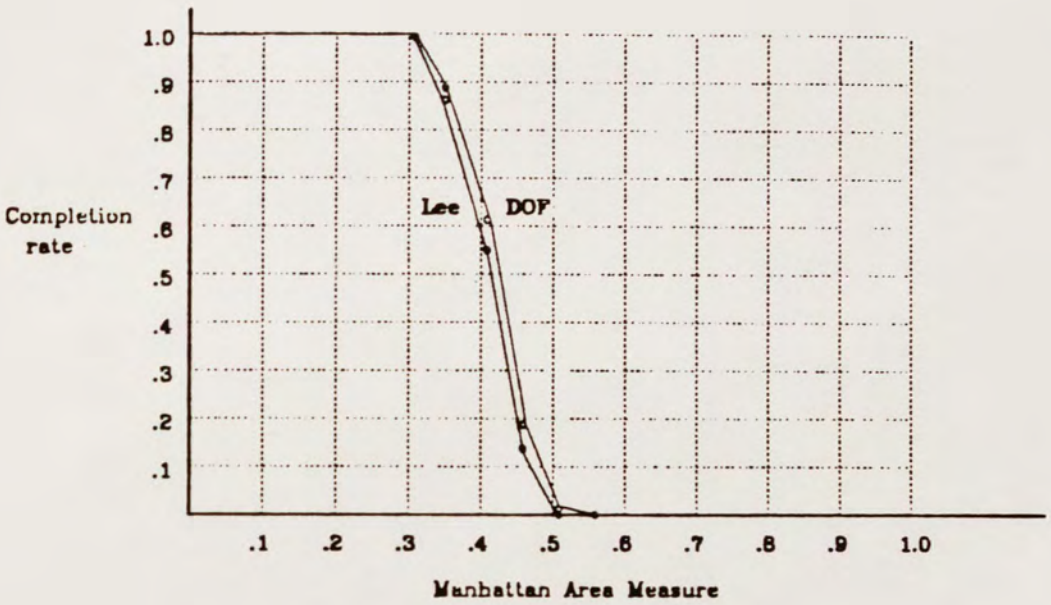


TABLE 19
LOW ASPECT RATIO AREAS WITH MULTIPOINT NETS:
DEGREE OF FREEDOM ALGORITHM

	MAM				
	.30	.35	.40	.45	.50
Comp. Rate	1.0	.86	.62	.18	.02
Polysilicon	94.88	113.93	131.29	144	134
Metal	245.22	279.72	316.81	346.11	412
Vias	22.96	27.11	31.51	34.44	37
0 via path	.005	.006	.007	.010	0.0
1 via path	.46	.455	.455	.450	.522
2 via path	.51	.517	.486	.470	.435
3 via path	.009	.009	.027	.037	.000
4 via path	.004	.011	.018	.018	.000
5+ via path	.001	.001	.004	.015	0043

TABLE 20
LOW ASPECT RATIO AREAS WITH MULTIPOINT NETS:
LEE ALGORITHM

	MAM				
	.30	.35	.40	.45	.50
Comp. Rate	1.0	.88	.56	.14	0.0
Polysilicon	95.6	115.07	132.57	137	0.0
Metal	244.6	279.14	315.61	345.0	0.0
Vias	23.56	28.30	33.04	36.86	0.0

APPENDIX C

ALCP Routing Results

This appendix contains the raw data for the area routing statistics for the ALCP algorithm presented in the future work session of the dissertation. The following statistics have been compliled:

Completion rate:	the percentage of areas that the algorithm completed for a given MAM value.
Polysilicon:	the mean length of polysilicon wire.
Metal:	the mean length of metal wire.
Vias:	the mean number of vias.
DOF Vias:	the mean percentage of paths that were routed using 0,1,2, etc vias.
Density:	the density of the area. The method of computation is given in Chapter 6.

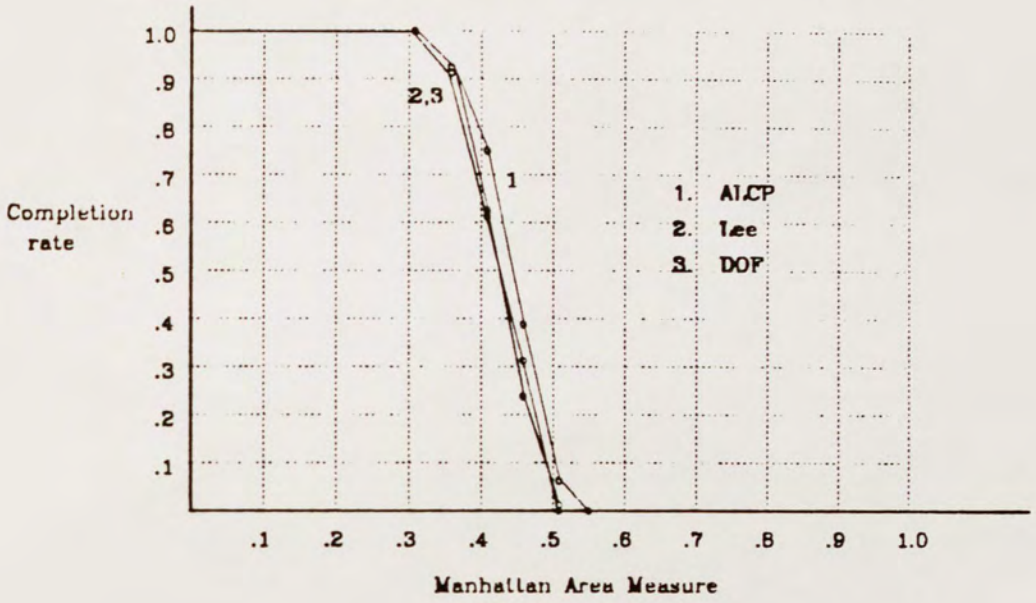


TABLE 21
ALCP COMPARISON
LOW ASPECT RATIO AREAS WITH TWO POINT NETS:
DEGREE OF FREEDOM ALGORITHM

	MAM				
	.30	.35	.40	.45	.50
Comp. Rate	1.0	.92	.62	.32	0.0
Polysilicon	87.64	105.72	123	145	0.0
Metal	231.46	265.33	305	326	0.0
Vias	20.4	24.24	28	35	0.0
0 via path	.008	.01	.012	.007	0.0
1 via path	.299	.29	.296	.274	0.0
2 via path	.661	.66	.628	.576	0.0
3 via path	.026	.024	.031	.078	0.0
4 via path	.006	.014	.032	.056	0.0
5+ via path	0.0	.001	.002	.007	0.0

TABLE 22
ALCP COMPARISON
LOW ASPECT RATIO AREAS WITH TWO POINT NETS:
LEE ALGORITHM

	MAM				
	.30	.35	.40	.45	.50
Comp. Rate	1.0	.94	.64	.24	.02
Polysilicon	88.16	107.74	128.5	152	189
Metal	231.42	264.7	303	321	368
Vias	20.8	25.4	30.62	38	51

TABLE 23
ALCP COMPARISON
LOW ASPECT RATIO AREAS WITH TWO POINT NETS:
ALCP ALGORITHM

	MAM				
	.30	.35	.40	.45	.50
Comp. Rate	1.0	.94	.76	.38	.06
Density	7.22	8.72	9.8	10.52	11.0
Polysilicon	106.74	122.78	141	160	148
Metal	231.42	264.87	303	325	368
Vias	20.68	24.34	28	33	34

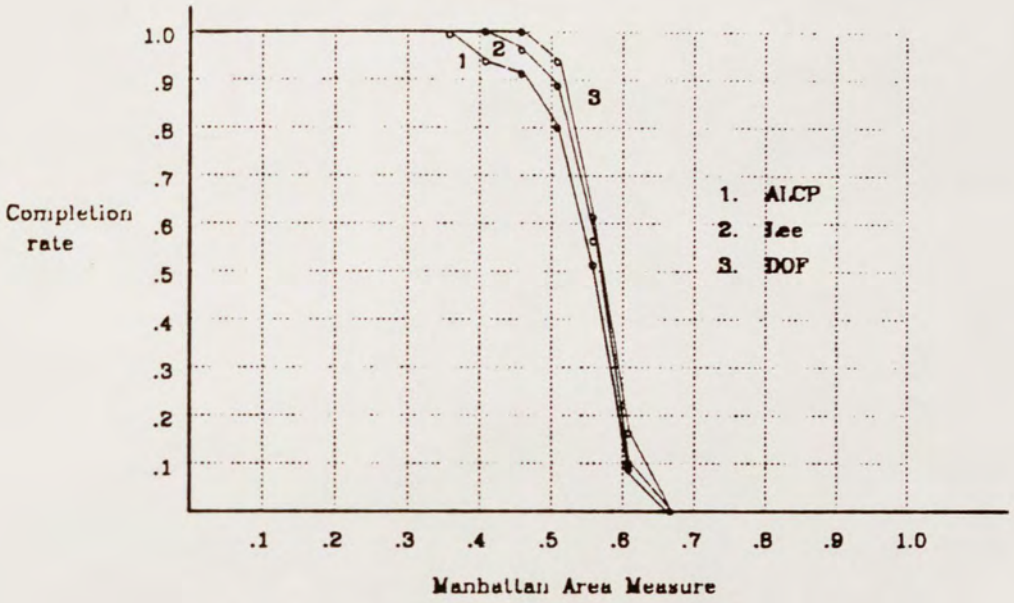


TABLE 24
ALCP COMPARISON
MODERATE ASPECT RATIO AREAS WITH TWO POINT NETS:
DEGREE OF FREEDOM ALGORITHM

	.40	.45	.50	.55	60
Comp. Rate	1.0	1.0	.94	.62	.16
Polysilicon	280	318.22	357	398.8	445
Metal	391	441.32	487	525.3	445
Vias	37	43.2	49	54.9	62
0 via path	.008	.0077	.007	.0083	.010
1 via path	.447	.429	.416	.419	.419
2 via path	.435	.422	.409	.401	.383
3 via path	.110	.134	.153	.144	.154
4 via path	.002	0.0	.011	.019	.011
5+ via path	0.0	.0007	.011	006	.021

TABLE 25
ALCP COMPARISON
MODERATE ASPECT RATIO AREAS WITH TWO POINT NETS:
LEE ALGORITHM

	MAM				
	.40	.45	.50	.55	60
Comp. Rate	1.0	.96	.88	.56	.08
Polysilicon	285	327.23	370	419.9	463
Metal	391	438.56	487	524.21	575
Vias	39.3	45.85	54	62.79	70

TABLE 26
ALCP COMPARISON
MODERATE ASPECT RATIO AREAS WITH TWO POINT NETS:
ACLP ALGORITHM

	MAM				
	.40	.45	.50	.55	.60
Comp. Rate	.94	.92	.80	.52	.10
Density	13.46	14.88	16.34	17.74	18.6
Polysilicon	340	372.5	410.8	441.88	473
Metal	390	436.11	490	531.96	549
Vias	39	44.69	51	56.46	62

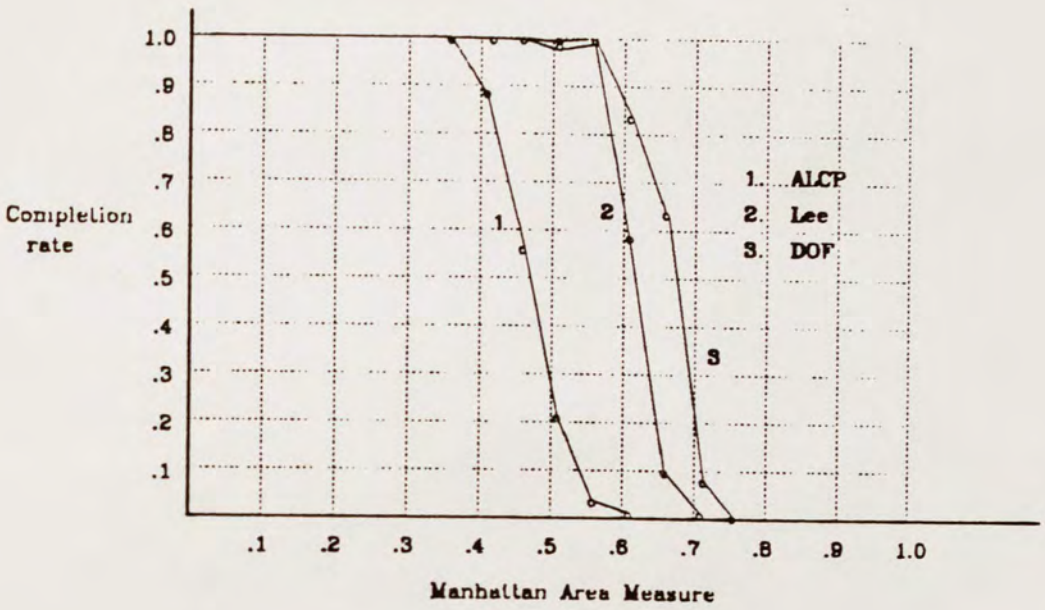


TABLE 27
HIGH ASPECT RATIO AREAS WITH TWO POINT NETS:
DEGREE OF FREEDOM ALGORITHM

	MAM							
	.35	.40	.45	.50	.55	.60	.65	.70
Comp. Rate	1.0	1.0	1.0	1.0	1.0	.84	.64	.08
Polysilicon	345.8	39	445.1	491	545.8	598	648.7	682.5
Metal	302.5	349	392.4	436	481.2	527	574.81	614
Vias	31.22	37	42.6	53	55.52	61	68.28	68.75
0 via path	.003	.003	.002	.002	.002	.001	.002	.006
1 via path	.643	.626	.601	.583	.560	.538	.533	.534
2 via path	.218	.213	.205	.201	.192	.188	.175	.194
3 via path	.136	.158	.191	.214	.244	.265	.271	.243
4 via path	0.0	0.0	0.0	0.0	.002	.003	.006	0.0
5+ via path	0.0	0.0	0.0	0.0	.001	.004	.013	.024

TABLE 28
HIGH ASPECT RATIO AREAS WITH TWO POINT NETS:
LEE ALGORITHM

	MAM							
	.35	.40	.45	.50	.55	.60	.65	.70
Comp. Rate	1.0	1.0	1.0	.98	1.0	.58	.10	0.0
Polysilicon	347.7	398	451.8	505	564.1	613	640.6	0.0
Metal	303.58	352	396.32	444.7	501.5	553	592	0.0
Vias	33.5	39	45.84	53	63.8	71	71.4	0.0

TABLE 29
HIGH ASPECT RATIO AREAS WITH TWO POINT NETS:
ALCP ALGORITHM

	MAM							
	.35	.40	.45	.50	.55	.60	.65	.70
Comp. Rate	.98	.88	.56	.22	.04	.02	0.0	0.0
Density	12.92	14.6	16.22	17.6	18.06	18.32	22.74	
Polysilicon	401.1	448	509.4	536	546.5	597	0.0	0.0
Metal	301.45	347	388.82	426	438	533	0.0	0.0
Vias	33.65	40	43.86	47	45	53	0.0	

LIST OF REFERENCES

Ag77.

Agrawal, P. "Routing of Printed Circuit Cards: Density Analysis and Routing Algorithms." USCEE Report 495, University of Southern California, June 1977.

Ak67.

Akers, S. B. "A Modification of Lee's Path Connection Algorithm." *IEEE Transactions on Electronic Computers* EC16 (February 1967) : 97-98.

Ba81.

Baratz, A. "Algorithms for Integrated Circuit Routing." Ph.D. dissertation, M.I.T., August 1981.

BuPe83.

Burstein, M. and Pelavin, R. "Hierarchical Channel Router." *Integration* 1 (1983) : 21-38.

ChHsKu83.

Chen, N. P.; Hsu, C. P.; and Kuh, E. S. "The Berkley Building-Block (BBL) Layout System for VLSI Design." *Proceedings of the IFIP International Conference on VLSI*, Trondheim, Norway, pp. 37-44, North-Holland, August 1983.

De76.

Deutsch, D. N. "A Dogleg Channel Router." *Proceedings of the Thirteenth Design Automation Conference*, pp. 425-433, San Fransisco: ACM and IEEE, June 1976.

Di59.

Dijkstra, E. W. "A Note on Two Problems in Connecton with Graphs." *Numerische Mathematik* 1 (1959): 285-292.

GaJo79.

Garey, M. R. and Johnson, D. S. *Computers and Intractability*. San Francisco, Ca: W. H. Freeman and Company, 1979.

Ga78.

Gavril, F. "Algorithms for Miminum Coloring Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph." *SIAM Journal of Computing* 1 (June 1978) : 180-187.

HaSt71.
Hashimoto, A. and Stevens, S. "Wire Routing by Optimizing Channel Assignment Within Large Apertures." *Proceedings of the Eighth Design Automation Conference*, pp. 155-169, Atlantic City, NJ : ACM and IEEE, 1971.

HaSaBe80.
Hayns, W.; Sansen W.; and Beke, H. "A Linc Expansion Algorithm for the General Routing Problem with a Guaranteed Solution." *Proceedings of the 17th Design Automation Conference*, pp. 243-249, Minneapolis, Min.: ACM and IEEE, June 1980.

Hi69.
Hightower, D. "A Solution to Line-Routing Problems on the Continuous Plane." *Proceedings of the Design Automation Workshop*, pp. 1-24, 1969.

KeRi78.
Kernighan, B. W. and Ritchey, D. M. *The C Programming Language*. Englewood Cliffs, NJ : Prentice-Hall, 1978.

Ko82.
Korn, R. K. "An Efficient Variable-Cost Maze Router." *Proceedings of the 19th Design Automation Conference*, pp. 425-431, Las Vegas: ACM and IEEE, June 1982.

La80.
LaPaugh, A. "Algorithms for Integrated Circuit Layout: An Analytic Approach." Ph.D. dissertation, MIT, December 1980.

Le61.
Lee, C. Y. "An Algorithm for Path Connection and Its Applications." *IRE Transactions on Electronic Computers* (September 1961) : 346-365.

Mi70.
Mikami, K. and Tabushi, K. "A Computer Program for Optimal Routing of Printed Circuit Connectors." *IFIPS*, pp. 1475-1478, 1970.

Mo59.
Moore, E. F. "The Shortest Path Through a Maze." *Proceedings of the Iternational Symposium on Switching Theory*, pp. 285-292, Harvard Univ. Press, 1959.

PeDeSc76.
Persky, G.; Deutsch, D.; and Schweikert, D. "LTX - A System for the Directed Automatic Design of LSI Circuits." *Proceedings of the 13th Design Automation Conference*, San Fransisco: ACM and IEEE, June 1976.

Pi81.

Pinter, R. Y. "Optimum Routing in Rectilinear Channels." *CMU Conference on VLSI Systems and Computations*, pp. 160-177, Rockwell, MD : Computer Science Press, October 1981.

Ri82a.

Rivest, R. L. "The "PI" (Placement and Interconnect) System." *Proceedings of the 19th Design Automation Conference*, pp. 475-481, Las Vegas: ACM and IEEE, June 1982.

Ri82b.

Rivest, R. L. Private Communication, 1981.

RiFi82.

Rivest, R. L. and Fiduccia, C. M. "A Greedy Channel Router." *Proceedings of the 19th Design Automation Conference*, pp. 418-424, Las Vegas: ACM and IEEE, June 1982.

RiBaMi81.

Rivest, R. L.; Baratz, A. E.; and Miller, G. "Provably Good Channel Routing Algorithms." *CMU Conference on VLSI Systems and Computations*, pp. 153-159, Rockwell, MD : Computer Science Press, October 1981.

Ru74.

Rubin, F. "The Lee Connection Algorithm." *IEEE Transactions on Computers* C-23 (1974) : 907-914.

Sm83.

Smith, L. "An Analysis of Area Routing." Ph.D. dissertation, Stanford, CA., February 1983.

So78.

Soukup, J. "Fast Maze Router." *Proceedings of the 15th Design Automation Conference*, pp. 100-102, Las Vegas : ACM and IEEE, June 1978.

Su82.

Supowit, K. J. "A Minimum Impact Routing Algorithm" *Proceedings of the 19th Design Automation Conference*, pp. 104-112, Las Vegas : ACM and IEEE, June 1982.

WeMo81.

Weinreb, D. and Moon D. "Lisp Machine Manual", MIT Artificial Intelligence Laboratory, 1981.

YoKu82.

Yoshimura, T. and Kuh, E. S. "Efficient Algorithms for Channel Routing." *IEEE Transactions on CAD of Integrated Circuits and Systems* CAD-1 (January 1982) : 25-35.

Za83.

Zahir, S. Private Communication, 1983.