GENERAL RIVER ROUTING ALGORITHM*

Chi-Ping Hsu

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

A general and practical river routing algorithm is described. It is assumed that there is one layer for routing and terminals are on the boundaries of an arbitrarily shaped rectilinear routing region. All nets are two-terminal nets with pre-assigned (may be different) widths and no crossover between nets is allowed. The minimum separation between the edges of two adjacent wires is input as the design rule. This algorithm assumes no grid on the plane and will always generate a solution if a solution exists. The number of corners is reduced by flipping of corners. An analysis to determine the minimum space required for a strait-type river routing problem is included.

Let B be the number of boundary segments and T be the total number of terminals. The time complexity is of $O(T(B+T)^2)$ and the storage required is $O((B+T)^2)$. This algorithm is implemented as part of the design station under development at the University of California, Berkeley.

1. Introduction

With the advent of VLSI technologies, the complexity of the circuits on a single chip has increased drastically. Microprocessors and many digital signal processors are now built on a single chip. These processors usually have sets of data busses which interconnect different circuit blocks on the chip. At the chip planning stage, the designer can, in general, determine the sequence of the input and output busses of each block, so that every block has the output busses ordered in the same sequence as that of the input busses of the receiving circuit blocks. Since the input and output busses of the blocks have the same sequence, it is possible to make the interconnections between blocks on a single layer. The problem of interconnecting pairs of pins in two rows with the same sequence on a single layer is usually referred to as the "river routing problem"[1,2,3,4].

This paper presents a routing algorithm which handles a much more general and practical river routing problem. This algorithm can handle arbitrarily shaped rectilinear routing regions with nets for which wire widths can be defined independently. Also, it is a gridless routing algorithm. The minimum separation between the edges of two wires is input as a parameter. This algorithm guarantees that a solution can be found if one exists. The design rule check and an analysis to determine the minimum space needed for a strait-type river routing problem are included.

In section 2, we introduce some basic terminologies and define the river routing problems. In section 3, the algorithm for routing is presented. In section 4, the calculation of the minimum width for the strait-type problem is described.

2. Terminology and Problem Formulation

A *routing region* is a continuous area between circuit blocks that can be used for routing. It is specified by the boundaries of each available layer. (Note that each layer may have different boundaries). A *terminal* is either an input or output pin on the boundaries of a routing region. A terminal is characterized by its location, width, and the layer it is on. A *signal net* (or simply *net)* is a set of terminals to be interconnected by wires. A *routing segment* is a horizontal or vertical wire segment on a specified layer which implements all or part of a signal net. It is represented by its starting and ending points, the width of the segment, its associated layer, and the signal net it belongs to. A *contact* is an area where a set of layers are electrically interconnected. It has a geometry representing its bounding box and a set of associated layers.

A general *routing problem* consists of a routing region, a set of signal net definitions, and a set of design rule parameters. A *solution* to a routing problem is a set of routing segments and contacts inside the routing region, which implements the set of signal nets definitions without design rule violation.

A general *river routing problem* is a special case of the general routing problem where

(a) Only one layer is available for routing in the routing region.

(b) All terminals are on the same layer.

* This paper is supported in part by Air Force Office of Scientific Research, contract number F49620-79-C-0178, Joint Services Electronics Program, National Science Foundation Grant ECS-8201580, and Bell Laboratories at Murray Hill, New Jersey.

0738-100X/83/0000/0578$1.00 © 1983 IEEE

(c) Every signal net consists of exactly two terminals.

(d) Terminals are located in such a way that no crossover between signal nets is necessary for a solution to exist.

(e) The routing region has no internal blockage.

Fig. 1 shows an example of a general river routing problem.

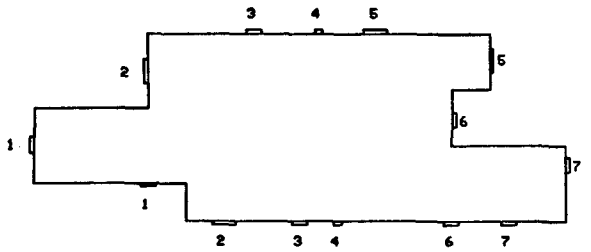

Fig.1 An example of the general river routing problem

A *strait-type river routing problem* is a special case of river routing problem. It is a river routing problem, where

(a) The routing region can be specified by two opposite boundary segment lists that are both monotonic in either X or Y direction.

(b) All terminals are on horizontal(vertical) boundary segments if the two boundary segment lists are monotonic in the X(Y) direction.

(c) Every signal net has one terminal on each of the two boundary segment lists.

A strait-type river routing problem is horizontal if the two boundaries are monotonic in X direction and vice versa. The separation between the two boundaries is usually refered to as the *width* of the strait. Fig. 2 is a horizontal strait-type river routing problem.

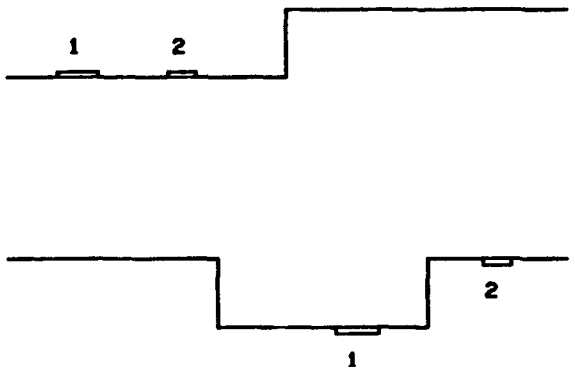

Fig.2 A strait-type river routing problem

3. General River Routing Algorithm

An ordered list of segments is *continuous* if the starting point of every segment, except for the first segment, is the ending point of the previous one. A *path* is a continuous list of alternating horizontal and vertical routing segments. A terminal connected with the first segment of a path will be called a *starting terminal* and the terminal connected with the last segment is an *ending terminal*. We assume that the widths of the routing segments of a signal net are the same and are predefined by the user.

This algorithm routes one net at a time. It starts with the assignment of the starting terminal for each net and the net order is determined by the sequence of terminals on the boundaries. Path searching is done by routing each net in turn as close to the boundaries as possible. Unnecessary corners are then removed by flipping of corners.

3.1. Starting Terminal Assignment

For a river routing problem, all nets are two-terminal nets. Without loss of generality, we assume that every path is counter-clockwise along the boundary. Every net has two possible paths *along the boundary*. Clearly, the two possible paths then correspond to the two possible choices of starting terminal for the net. Fig. 3 shows two possible paths along the boundary for a net {T1,T2}. Path P1 has T1 as its starting terminal and path P2 has T2 as its starting terminal.

The starting terminal for a net is chosen, independently of all other nets, such that the shorter path is selected. This is done by calculating the total length of the boundary segments counter-clockwise between the starting and ending terminals and compare it with half of the total length of the boundaries of the routing region. In fig. 3, path P1 is shorter than path P2. So, terminal T1 is assigned to be the starting terminal for the net.

3.2. Net Ordering

Since every path is counter-clockwise and begins at the starting terminal, there are constraints on the order of the nets to be routed. A net is routed only after all nets, which have one or more terminals on the counter-clockwise portion of the boundaries between the starting and ending terminals of the net, have been routed.

To determine the net order, we first generate a circular list of all terminals ordered in counter-clockwise direction according to their positions on the boundaries. A planarity check is performed to see whether there exist crossovers by using a stack. If so, then the input problem is not a river routing problem. Otherwise, we order the nets as follows:

NET-ORDERING

(1) stack ST = empty;
i = 1;
N = the total number of signal nets;
T = any terminal in the circular list;
Every starting terminal is marked as NOT PUSHED.
Every ending terminal is marked as NOT

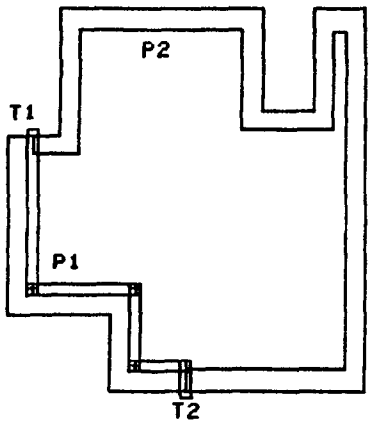

Fig.3 Two possible paths along the boundary, path P1 is shorter, so terminal T1 is assigned as the starting terminal.

```
    MATCHED.
(2) while i <= N
    begin
        if T is a starting terminal that is NOT
        PUSHED,
        begin
            push T on ST;
            mark T as PUSHED;
        end
        else
        begin
            if T is an ending terminal that is NOT
            MATCHED,
            begin
                if T and the top element of ST
                belong to the same net,
                begin
                    mark T as MATCHED;
                    pop the top element from ST;
                    assign the number i to the
                    net;
                    increment i by one;
                end;
            end;
        end;
        T = next terminal in the circular list;
    end;
```

Assert: The nets are ordered in increasing order of the assigned number.

Fig. 4 shows the starting terminal for each net and the net ordering is {1,7,6,5,4,2,3}.

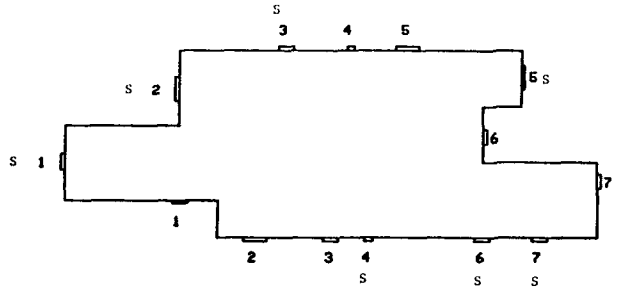

Fig.4 The starting terminal for each net and the net ordering {1,7,6,5,4,2,3}

3.3. Path Searching

For each net in turn, two continuous constraint segment lists are created by tracing through the boundary segments and the existing routing segments that are *exposed* to the current net. A path is then formed by routing counter-clockwise as close to one of the constraint list as possible, beginning at the starting terminal and stopping at the ending terminal of the net. The path is then checked against the other constraint list for design rule violation. If a violation occurs, it means that there is not enough space for any solution to exist, and from the topology of the current routing, the user can easily determine where the space should be added.

Fig 5 shows constraint lists 'abcd' and 'ihgf' for net A and the path created by routing as close to list 'abcd' as possible. The path is then checked against the list 'ihgf' for design rule violations.

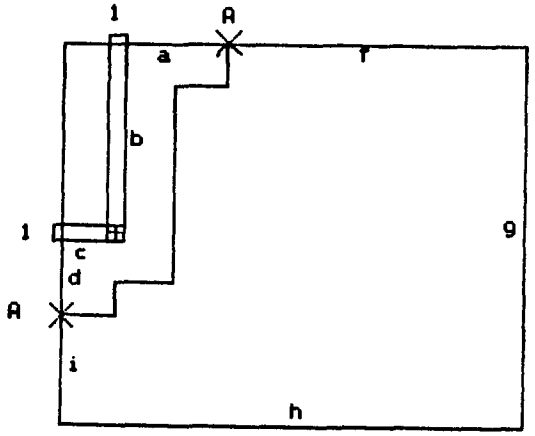

Fig.5 Constraint lists 'abcd' and 'fghi' for net A and the path created by routing as close to one of the constraint list as possible.

3.4. Corner Minimization

After we have done the path searching for all nets without design rule violation, we already have a solution. Every net has a unique path associated with it. All paths are pushed outward against the boundaries and the excess space remains in the center of the routing region. Fig 6 shows an example of the routing after path searching.

The corner minimization is done in a systematic way of flipping corners toward the inside of the routing region. We minimize the corners of one net at a time. The order of the nets for this operation is just the reverse of the order determined by the previous net ordering step. The net that is routed last is minimized first. Equivalently, we are minimizing the corners of the paths from inside nets toward the outside nets of the routing region.

Every corner of a path belongs to one of the eight possible cases as shown in Fig.7. Since every path is routed in the counter-clockwise direction, four of the cases can have their corners *flipped* toward the inside of the routing region. In fig.7, cases a,b,c,d can be transformed to cases e,f,g,h by

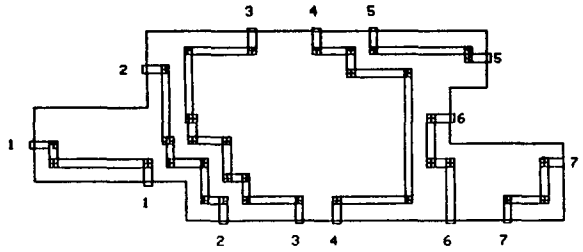

Fig.6 A boundary-packed solution after path searching

flipping toward the inside, where the inside is indicated by a black dot. A constraint segment list is generated by the same way as in the path searching step. Before we flip a corner, we first check whether the flipping will generate any design rule violation against the constraint list. If it does not have any design rule violation, we will flip the corner and thus eliminates two corners of the path. Otherwise, we skip this corner and check the next corner of the path. Fig 8 shows one path before and after flipping of corners.

3.5. Summary

In summary, this river routing algorithm routes all nets against the boundaries of the routing region and then tries to minimize the number of corners. The first step "starting terminal assignment" tries to select a shorter path for each net and spread the nets against all boundaries. It should be noted that if the problem can be routed without crossovers, this assignment will not cause any crossover to occur. Based on the assignment, the net order is determined by using a stack. Let T be the total number of terminals. The while loop in NET-ORDERING will be executed at most 2T times since at most two cycles around the circular terminal list is necessary if the problem is a river routing problem. Because of the assumption that all paths are counter-clockwise against the boundary, the paths are generated easily by going along the constraint list. Finally, the corner minimization is done in reverse net order by flipping the corners toward the inside of the routing region. This step efficiently generates a very desirable final layout due to the fact that all paths are distributed around the whole routing region and the flipping operation is very efficient.

3.6. Existence of A Solution

A solution of a river routing problem is ***boundary-packed*** if and only if no path can be replaced by another path which is no farther from the boundary anywhere. Given a routing solution with n nets. The routing region is divided into n+1 subregions. If we imagine that we stand inside one of the subregions, a unique boundary-packed solution can be obtained by pushing all the nets against the boundaries. For a legal "starting terminal assignment", it uniquely determines one subregion where we stand. The solution we get after path searching and before corner minimization is ***the*** unique boundary-packed solution associated with the "starting terminal assignment" we get in the first step of the algorithm.

Given a river routing problem, there always exists a legal "starting terminal assignment". For any legal "starting terminal assignment", there exists a unique boundary-packed solution if a solution exists. This algorithm first generates a legal "starting terminal assignment" and then tries to find the unique boundary-packed solution associated with the assignment. Clearly, if the algorithm

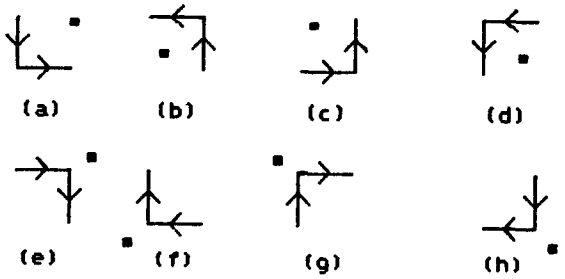

Fig.7 Eight possible cases of a corner

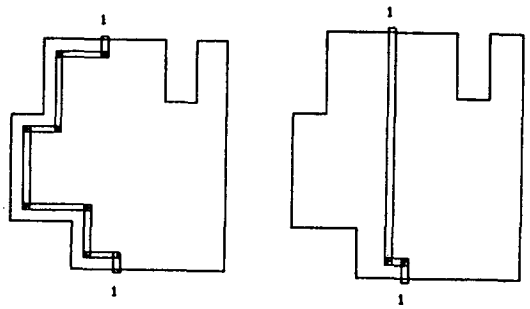

Fig.8 A path before and after flipping of corners

can not find the unique boundary-packed solution, it implies that no boundary-packed solution exists, which in turn implies that there is no solution.

4. Calculation of The Minimum Width For A Strait-type River Routing Problem

In the algorithm described above, we have implicitly assumed that the given routing region has a fixed topology. The algorithm will generate a solution if a solution exists, and returns a partially routed layout if there is no solution for the problem. If a solution exists, the excess space can be obtained by using the boundary-packed layout. However, if a solution does not exist, the information about the minimum additional space needed can only be partially obtained by using the incomplete layout.

For a strait-type river routing problem, the minimum width can be calculated exactly before we do the actual routing. The calculation starts with a pseudo-routing without design rule check and then the minimum width is determined to insure that no design rule violation will occur.

4.1. Pseudo-Routing Algorithm

Let us describe this algorithm for a horizontal strait-type river routing problem as shown in Fig. 2. As before a signal net for this type of problem con-

sists of two terminals. There, one is on the upper boundary and the other is on the lower boundary. Let Xui be the x coordinate of the terminal of signal net i on the upper boundary and Xli be the x coordinate of the terminal of signal net i on the lower boundary. Signal net i will be called a ***falling net*** if Xui < Xli, a ***trivial net*** if Xui = Xli, and a ***rising net*** if Xui > Xli.

Algorithm: Pseudo-routing

This algorithm proceeds net by net, in the order from left to right, generating a set of routing segments for each net in turn. Further, the set of routing segments for each net is generated by the simple procedure:

(1) If the net under consideration is a trivial net, we simply make a straight vertical routing segment connecting the two terminals and proceed to the next net. If the net is not a trivial net, we generate a *continuous* constraint segment list. This list of segments consists of the routing segments of the previous net just routed (or the left boundary segments if the net is the leftmost net) and portions of either the upper or lower (for rising or falling net) boundary segments between the previous net and terminal of the net.

(2) Generate a continuous list of segments with current net width by *licking* along the right edge of the constraint segments with the separation equal to the minimum spacing between adjacent wire edges. Note that boundary segments have width equal to zero.

(3) Generate two vertical routing segments from the two terminals of the net to the segment list obtained in step (2). Delete the segments to the left of the vertical cutline through the left terminal. The resulting continuous segment list is the routing segments of this net.

Fig. 9 shows the routing operation for net 2. The constraint segment list consists of the routing segments of net 1 and portions of the lower boundary between the two terminals of net 1 and net 2. A continuous list is generated by licking along the constraint list and the the resulting routing segment list is obtained for net 2.

Informally, we route one net at a time in the order from left to right. If the net under consideration is a falling net, we go downward from the left terminal of the net as far as we can, and then lick along the upper edge of the constraint segment list as close as possible until we reach the right terminal of the net. If the net is a rising net, we do the similar operation except we go upward instead of downward. No design rule check is performed against the opposite boundaries. If the net is trivial, we simply connect the two terminals directly by a straight vertical routing segment.

The intuition behind this algorithm is that no space to the left of the routing segments of a net can be used by the nets to its right. Since we route the nets from left to right, we would like to route a net in such a way that we leave the maximum available space to the right of the routing segments. Also, we have implicitly assumed that both the left and right boundaries consists of a single vertical segment. It is easy to see that, for each net, no routing segment needs to appear to the left of its left terminal. So, we route the nets by stacking all rising nets against the upper boundary, and all falling nets against the lower boundary. Fig. 10 shows the result of a test example.

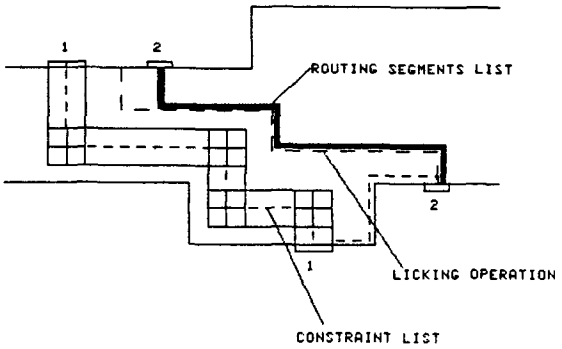

Fig.9 Constraint segment list, licking operation, and the resulting routing segments of a pseudo-routing for net 2

Note that this algorithm can handle arbitrarily shaped rectilinear routing region as long as the upper and lower boundaries are monotonic in the X direction.

If B is the number of boundary segments and T is the total number of terminals. The time complexity for this algorithm is $O(T(B+T)^2)$ and the storage required is $O((B+T)^2)$.

4.2. Calculation of The Minimum Width

The basic constraint on the width of the strait-type routing problem is that all routing segments must be inside the routing region, ie. between upper and lower boundaries. Since the pseudo-routing algorithm proceeds in such a way that all rising nets are stacked upward against the upper boundary and all falling nets are stacked downward against lower boundary. We can imagine that the upper boundary segments, and the rising

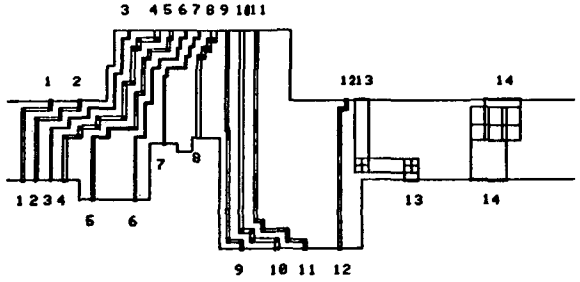

Fig.10 Pseudo-routing for a strait-type river routing problem

net segments (except for those vertical routing segments that connect to the lower boundary), forms a rigid body. Similarly, the lower boundary and the falling net segments (except for the routing segments that connect to the upper boundary), forms another rigid body. The concept is shown in Fig. 11. Now, we can put the two rigid bodies as close together as possible without design rule violation and calculate the *excess space* or the *minimum additional space* needed to have a solution. With the calculated minimum width, the result of the pseudo-routing *is* a solution to the problem. Since the general river routing algorithm guarantees that a solution can be found if one exists. We now can use the calculated width and do the actual routing by using the general river routing algorithm.

Let B be the number of boundary segments and T be the total number of terminals, the time complexity for this analysis is $O(B(B+T)^2)$.

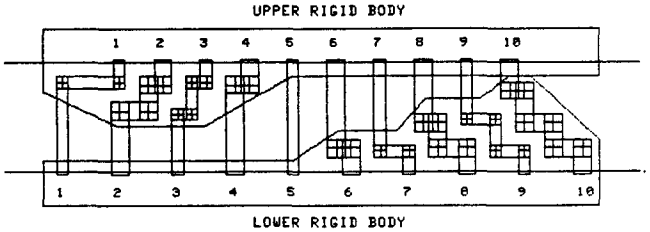

Fig.11 Formulation of the two rigid bodies

5. Experimental results

The two algorithms are implemented in the C language on a VAX-11/780 running the Berkeley Unix operating system, as part of the automatic placement and routing package of the design station under development at the University of California, Berkeley.

Fig. 12 shows the result of an example routed by the general river routing algorithm. There are 12 boundary segments and 14 terminals and the time required is 0.3 CPU seconds. Fig. 13 is a strait-type river routing problem with 22 boundary segments and 28 terminals. The analysis for the minimum width takes 0.2 seconds and the actual routing takes 1.0 seconds. Fig. 14 is a practical problem with 126 terminals and 12 boundary segments. The excess space has been detected and removed by using the boundary-packed solution. The time required is 7 seconds.

6. Conclusions

A general river routing algorithm is described. This algorithm can route signal nets with different widths inside an arbitrarily shaped routing region with one layer for routing. It guarantees that a solution can be found if one exists, and the solution is very close to the manual layout. The analysis to find the minimum width for a strait-type river routing problem guarantees that an optimal solution can be found. A simple generalization of the algorithm can handle a river routing problem with multiple-terminal nets.

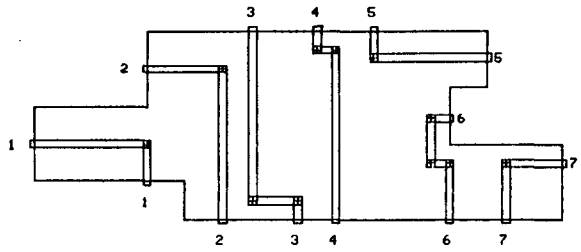

Fig.12 An example routed by the general river routing algorithm

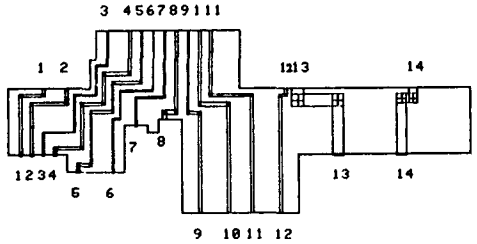

Fig.13 A strait-type river routing problem analyzed and routed

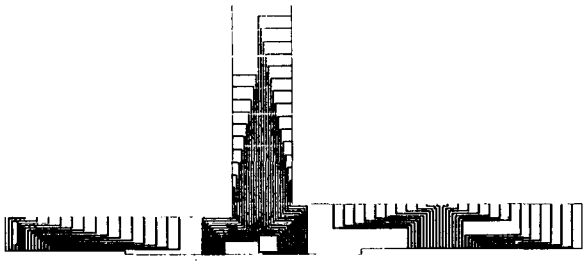

Fig. 14 A large and practical example

7. Acknowledgement

The author would like to thank professor E. S. Kuh for strong support and helpful advising, K. Keller who provides the convenient interactive graphics editor KIC, M. Takahashi for programming the interface with KIC, and the friendly users of professor R. Brodersen. Professor R. Newton and Professor A. Sangiovanni-Vincentelli have expedited this project.

8. References

[1] Baratz, A. E., "Algorithms for Integrated Circuit signal Routing" (Ph.D. dissertation), Dept. of Electrical Engineering and Computer Science, M.I.T., August 1981.

[2] Tompa, M., "An Optimal Solution to a Wire-Routing Problem", Proceedings of the twelfth Annual ACM Symposium on Theory of Computing, 1980, p161-176.

[3] Leiserson, C. E. ; Pinter, R. Y., "Optimal Placement for river Routing", Proceedings of the CMU Conference on VLSI systems and computations, October 1981.

[4] Pinter, R. Y., "River Routing: Methodology and Analysis", the third Caltech Conference on VLSI, March 21-23, 1983.