MOLE – A NEW TEMPLATE BASED ROUTER

by

Arvind Srinivasan

Memorandum No. UCB/ERL M89/58

17 May 1989

MOLE – A NEW TEMPLATE BASED ROUTER

by

Arvind Srinivasan

Memorandum No. UCB/ERL M89/58

17 May 1989

MOLE – A NEW TEMPLATE BASED ROUTER

by

Arvind Srinivasan

Memorandum No. UCB/ERL M89/58

17 May 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

This paper presents a new switchbox router - **Mole**. It uses a set of *patterns* or *templates* to perform connections between terminals sequentially. Routing is composed of combinations of a minimal set of carefully chosen patterns. A best set of patterns that realize a connection is selected, based on cost functions that minimize the number of jogs and vias.

The router makes use of information generated during the global routing phase in order to select pairs of terminals to be connected. Another novel feature of the router is a memory efficient way of implementing rip-up and rerouting. Each pattern has a unique identifying number and a characteristic parameter. For each connection that is made using the pattern, only the pattern number and the parameter are stored in memory.

The router considers blockages of arbitrary rectilinear shape within the switchbox and pins at any location. It is very fast and has successfully routed even the pathological Burstein's More Difficult switchbox.

1 Introduction

The switchbox routing problem is important in VLSI layout. The layout generation process is usually composed three phases - *Placement*, *Global-routing* and *Detailed routing*. The global routing phase divides the routing area into rectilinear blocks. Depending on the technology, two of the most commonly used regions are *channels* and *switchboxes*. A *channel* is a rectilinear routing region in which the terminals to be connected lie only on two sides of the region. In a *switchbox*, the terminals may lie on all four sides. It is generally considered that the switchbox routing problem is more difficult than the channel routing problem.

There have been several very good switchbox routers reported in the literature recently. **Weaver** [3] is a knowledge-based router that claims 100 percent routing. However, its long runtimes make its use in VLSI layout prohibitive. **Beaver** [2] is a new fast switchbox router that uses computational geometry techniques and produces excellent results. **Mighty** [6] is another very successful switchbox router that relies heavily on rip-up and rerouting to complete the nets.

This paper contributes some new heuristic methods to tackle the problem of switchbox routing. Additionally, some extensions to *general area* routing, in which the terminals are not constrained to be on the boundary of the routing region also proposed. The router uses a *control structure* similar to **Beaver** [2]. We observe that the control structure provides a global view of the contention between nets for the tracks and guides the routing process so that congestion is avoided towards the end.

The router consists of two phases of operation. In the first phase, the pattern router is used to make as many connections as possible. After this, a maze router is used to connect the remaining nets. If some nets still remain after the maze routing phase, rip-up and reroute is attempted. On most examples we have tested, rip-up and reroute was not necessary to complete the switchbox.

Mole is being developed as part of a Sea-of-gates layout project at U.C.Berkeley. The system

uses **Proud-II** [9] as the placement algorithm, **SOGR** [8] as the global router and **Mole** is envisaged as the detailed router.

2 Problem Definition

After the global routing phase, the chip is partitioned by hierarchical cuts in the horizontal and vertical direction into *blocks* as shown in Fig.1. Each block is defined by a rectangular box. There are terminals on the boundary as well as within the block, to which connections must be made. It is assumed that there are two layers on which detailed routing may be performed. Within each block there may be rectangular regions over which no routing can be done (blockages). An arbitrary rectilinear blockage is broken up into a set of rectangular blockages for the purposes of this discussion.

A *net* is defined to be a set of terminals which must be electrically connected together. The result of the global routing is a *tree* for each net. The vertices of the tree are the terminals. The tree is usually a minimum-cost tree based on some global-routing specific parmeters like congestion, track usage etc. The tree may be split into several sub-trees inside a block due to partitioning as shown in Fig. 1. The goal of the detailed router is to electrically connect all the terminals of each given net such that two different nets are not connected together.

3 Control Structure

Following up on the *control* idea generated by **Beaver** [2], we have generalized the concept to include terminals at arbitrary locations. On each layer, routing is done on one direction only. Terminals *control* a region around them which extends for a fixed distance in the preferred routing direction. Initially, this distance is set to a fraction f of the width of the block in the preferred

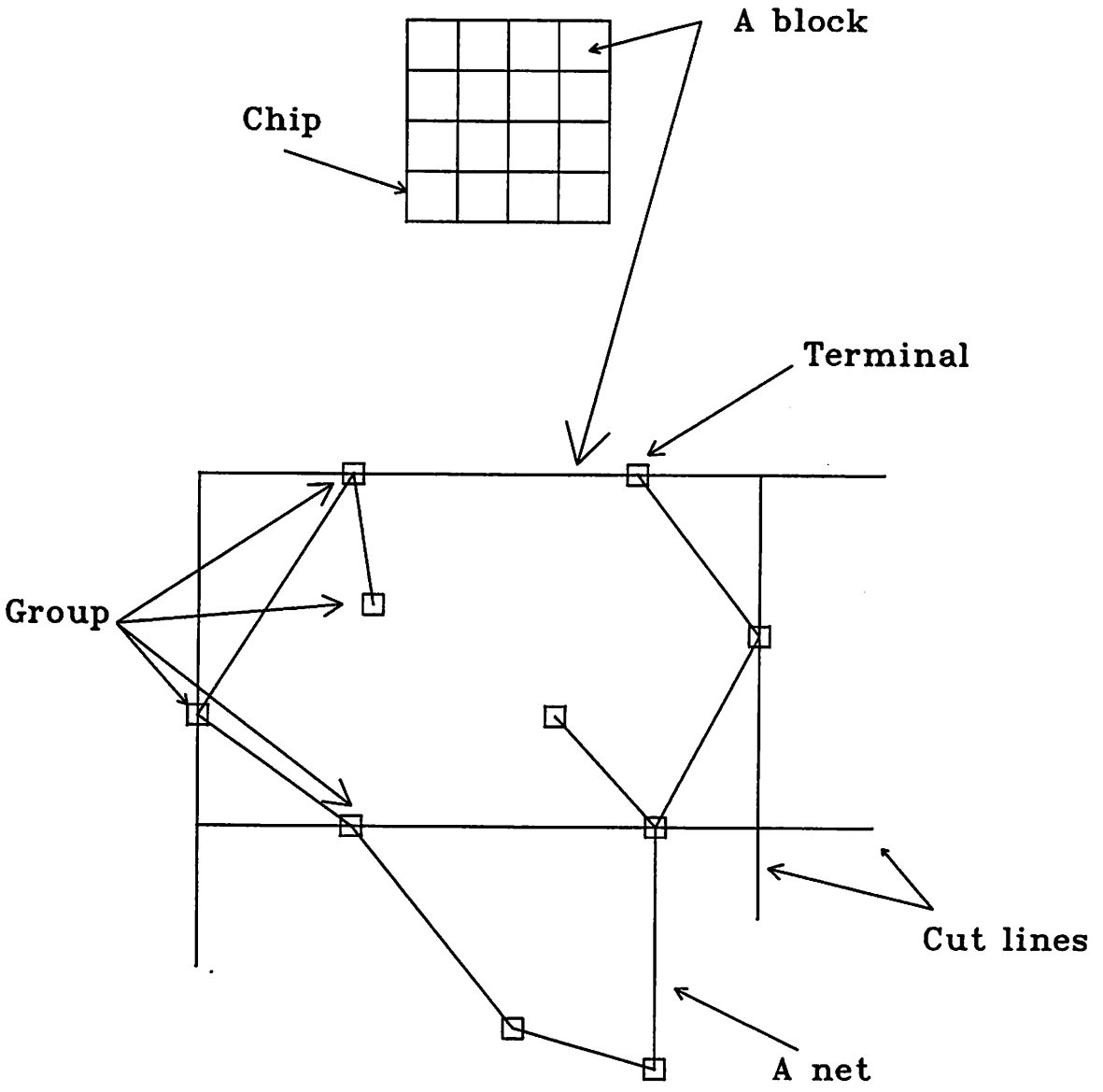


Figure 1: Blocks, groups and nets

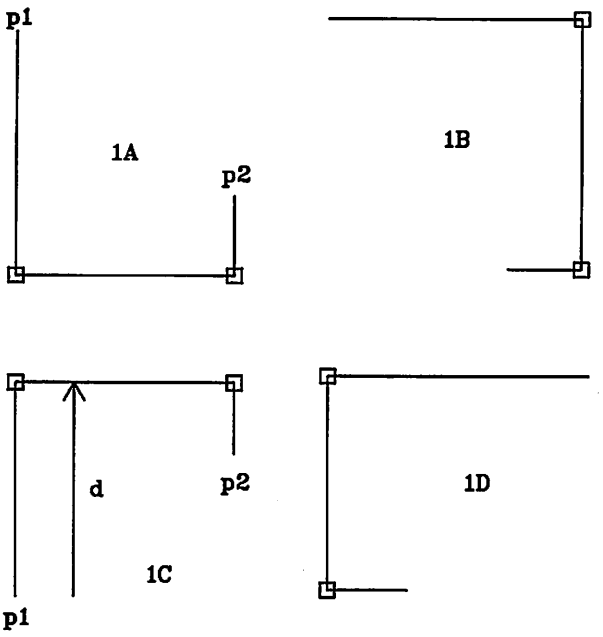
routing direction, for terminals on the edge of the block. For terminals not on the edge of the block, the control is extended in the preferred routing directions, for a distance equal to $f/6$ of the width of the block on both sides of the terminal. Within a block a net consists of several sub-trees which span the terminals of that net. We call the sub-trees *groups*. The parameter f is user specified, usually equal to 2/3.

4 Templates

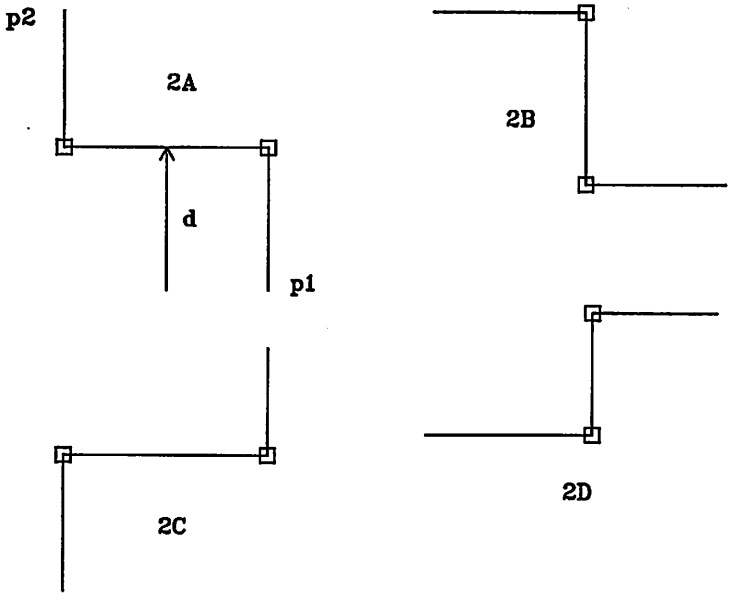
All routing consists of combinations of certain basic templates. We define four *basis* template *types* as shown in Fig. 2 and Fig. 3. Note that templates of type-3 are one the same layer and have no vias.

Within each type of basis template we have different templates depending on the orientation of the template. All the segments maybe identified as one, two or three-segment templates depending on the number of line-segments in the template. For the three-segment template, we define "central" and "outer" segments as shown in Fig. 4a. We do not use L-shaped patterns since these are only special cases of templates $1A, 1B, 1C$ or $1D$ with one of the outer segments being zero length (Fig. 4b).

Each template is assigned a unique pattern number and has a *characteristic* dimension d (this will be used later for efficient rip-up and reroute). All actual routing is composed of combinations of these patterns. As a result, any connection between two points which uses any one of these templates is completely characterized by the template number and its characteristic dimension.

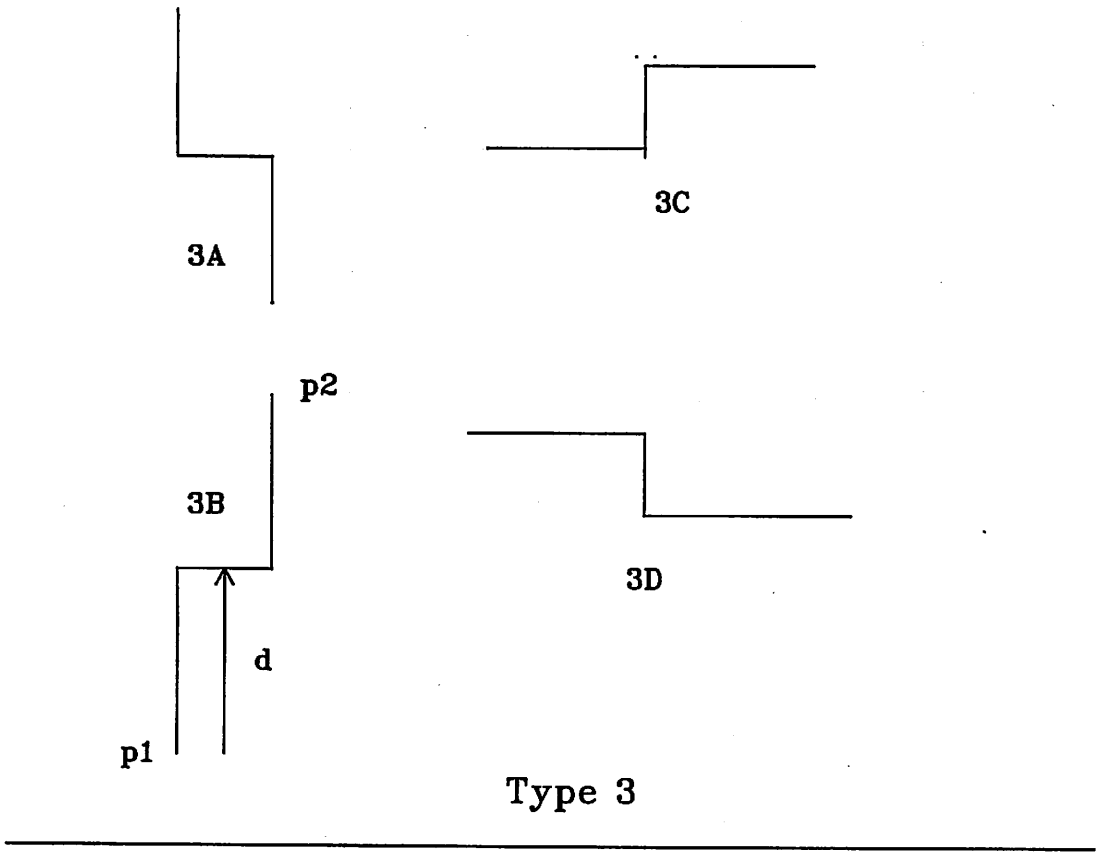


Type 1



Type 2

Figure 2: The template set



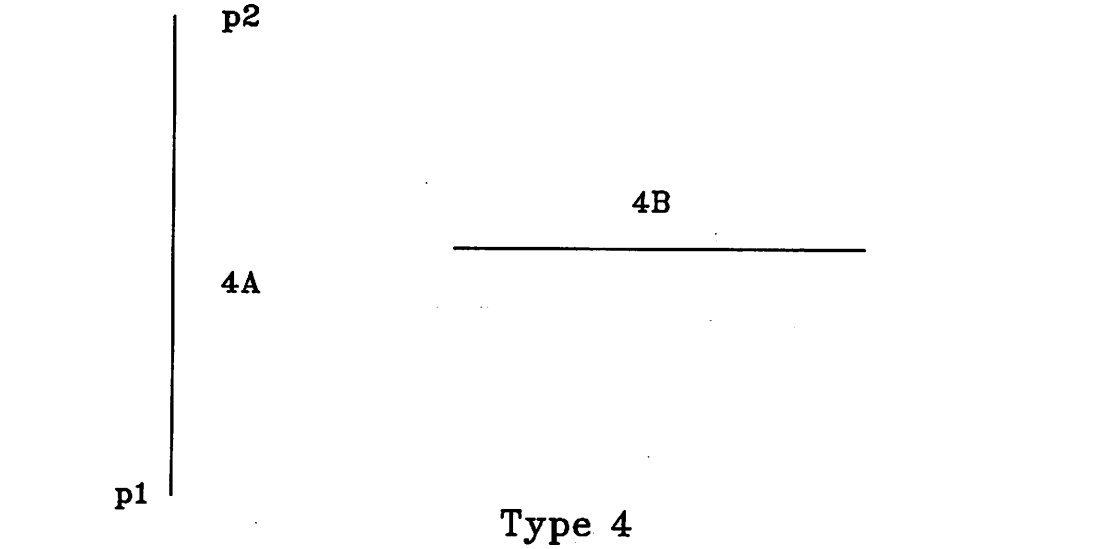
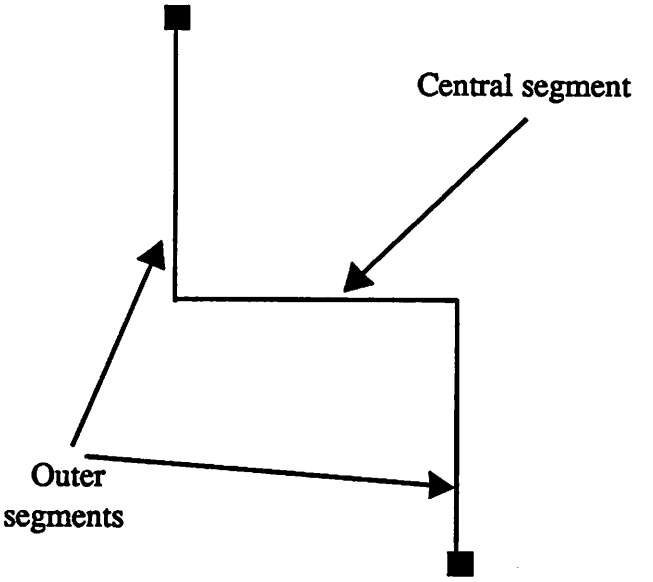
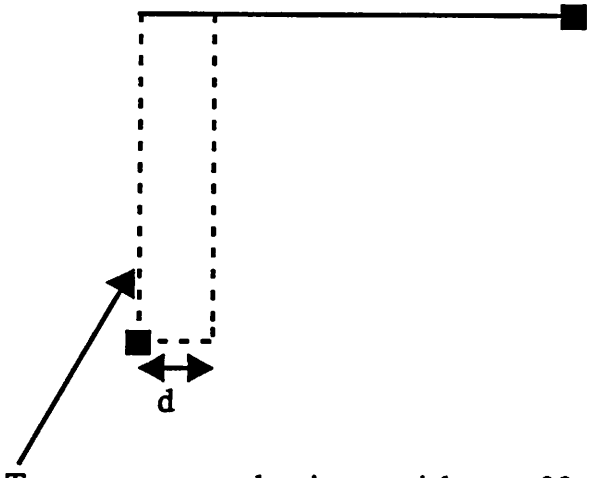


Figure 3: Templates continued



(a)



Two segment template is a special case of 3 segment, if d=0

(b)

Fig 4.

Figure 4: Some definitions

5 Routing Process

Consider a net N with terminals $\{t_1, t_2, \ldots, t_n\}$. The terminals are spanned by groups $g_1, g_2, \ldots, g_G$ (Fig. 1). The router connects terminals in pairs. Given n terminals, there are $\{n(n-1)/2\}$ possible pairs of terminals.

However, we make use of the tree generated during the global routing phase, since this tree is generated by minimizing an objective function that includes congestion information. We feel that this is another new feature of the detailed router. Conventional routers do not consider this important information provided by the global router.

There is an edge between two vertices in each group if there exists a global routing connection between the terminals corresponding to that group. The router assigns a weight to each edge. The weight is directly proportional to the half perimeter length of the edge. The edge with the highest weight is considered first. The terminals to be connected are taken in pairs, only if an edge exists in the the tree between that pair of terminals.

From the topology of the points, it is easy to determine a subset of all the templates which might be used to make a connection between two given points. Let the subset be $S = \{s_1, s_2, \ldots, s_k\}$. For each template s_i in S, a routing cost is evaluated according to the following scheme:- Let

C_v be the cost of a via,

C_c the cost of routing on a grid point controlled by a net other than itself,

C_b the cost of routing on a grid point that is blocked,

C_f the cost of routing over an empty grid point, and,

C_s the cost of routing on a grid point controlled by itself.

Then we set $C_b = \infty$, and $C_c \gg C_v > C_f > C_s$. For each connection between two terminals, the costs of the various possible template are evaluated. The template with the minimum cost is selected if its cost is less than C_c. If such a template is found then it is clearly seen that the connection

between the two points is uniquely identified by the template number chosen and the characteristic dimension of that template. These parameters are stored in memory (i.e remembered). Thus, if at some later stage in the routing process, a certain connection between two terminals needs to be "ripped" then only the template number and its characteristic dimension needs to be retrieved.

6 Recursive use of templates

If no template has a cost less than C_c then either that template is blocked by other nets or blockages, or the template passes over a region that is being controlled by another net. This case is a little complex and needs to be solved by some heuristics. We adopt the following strategy.

Please note that the templates consist of three, two or one line segments. The two-segment templates arise from type-1 when one section of that template is of zero length. We consider the following cases separately.

6.1 Three segment templates

In each template, either one or two segments can get blocked.

6.1.1 Case I - both outer segments blocked

In this case, we select the points on the outer line segments, closest to the blockages and try to route between them using the previously defined routing process recursively. The depth of recursion is a user specified parameter and for the sake of speed, usually one or two levels are recommended.

6.1.2 Case II - One outer segment and one central blocked

In this case, the point on the blocked segment and several points on the central segment are tried. Again, the recursion level is usually one or two levels deep.

6.1.3 Case III - Central segment blocked

In this case, several points on the two outer segments are chosen and routing is attempted on those. All these processes are illustrated in Fig. 5.

6.2 Two segment templates

Two cases arise - one segment blocked or two segments blocked.

6.2.1 Blockage on one segment

A succession of points are selected, starting away from the blockage and connection is tried to successive points on the other segment.

6.2.2 Blockage on both segments

In this case, we select the points on the two line segments, closest to the blockages and try to route them using the routing process recursively. The depth of recursion is a user specified parameter and for the sake of speed, usually one or two levels are recommended.

6.3 Single segment template

In this case, we select the farthest unblocked points from the two points to be connected and try to recursively connect them.

The routing process is repeated for all the nets. After one pass, some nets may be completed while others may be untouched. The regions over which the completed nets had control are now given to the closest terminals in the preferred routing direction. We re-prioritize the edges so that the edges belonging to untouched nets get maximum priority now.

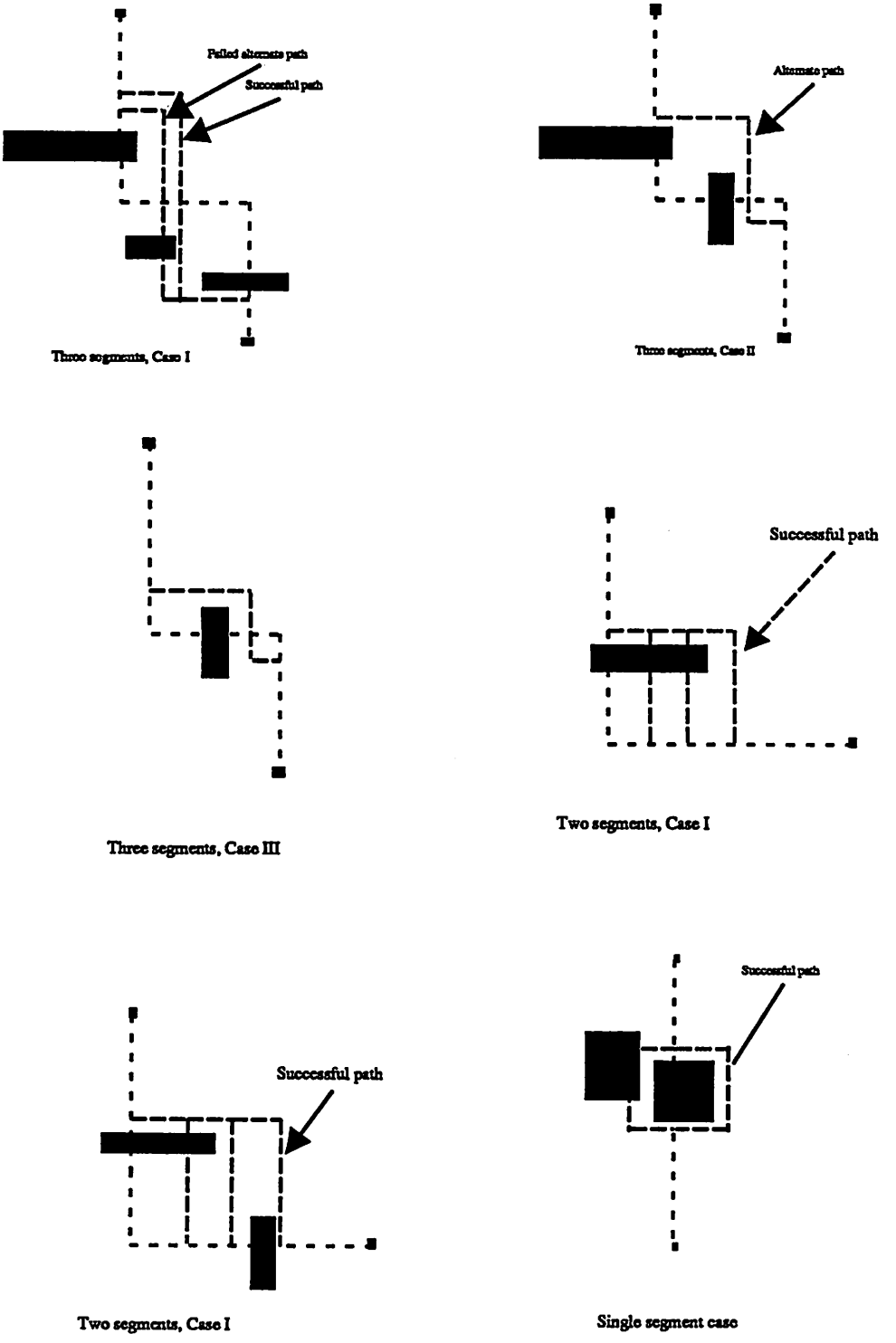


Figure 5: Avoiding blockages in the path

7 Dynamic update of control

It is possible that there are cases where due to the control, one net does not permit another net to be routed and a deadlock situation occurs. To avoid this, at every pass through the edges, we decrease the extent of control of the incomplete nets by a fixed amount. Thus eventually, the control gets reduced to zero towards the end of the routing, when maximum congestion starts occuring.

8 Rip-up and Maze Routing

For every connection between a pair of points, we store the template used and the characteristic dimension. Therefore the entire routing is flexible and the routes are stored in a very compact and memory-efficient way.

At the end of the template routing phase, some nets may still be unconnected. However, we have a feasible solution for the other nets. In the *rip-up* phase, an attempt is made to route the remaining unconnected nets by modifying the solution found so far, without reducing the number of nets completed.

First, we place all the edges for which successful connections were found in a *rip-up list.* For each edge in the rip-up list, we rip-up the existing connection, and attempt to reroute it using a different set of templates from the one used before. We also delete that edge from the rip-up list. If a different connection is found, then we have found an alternative route for a particular pair of terminals, possibly freeing up some blocked nets. So, we try to route the unconnected nets by invoking a maze router on them. If no new connection is found, we restore the ripped up connection and continue the above process on the remaining edges in the rip-up list. This process is repeated until no edges remain in the rip-up list or all the nets are completed.

9 Algorithm

```
algorithm route(netlist) {
    read global routing tree information;
    form net connection graph G;
    for each edge e in G {
        calculate weight of each edge;
        push edge onto priority list L;
    }
    do {
        for each edge e in L {
            success = template_route(e);
            if(success) then S = S + e;
        }
    }
    while((L not empty) and (success));
    if nets still remain {
        for each edge e in S {
            try alternate template for e;
            try maze routing of remaining nets;
        }
    }
}

procedure template_route(e) {
```

```
    if(level of recursion > max_levels) return(failure);
    S = select_possible_templates(e);
    min = min_cost(S);
    if(min < blocked_cost) {
        implement(e);
    } else {
        (e1, e2) = split(e)
        template_route(e1);
        template_route(e2);
    }
    return(success);
}
```

10 Results

We have implemented a preliminary version of the router in about 7500 lines of C code and the results are encouraging. Several examples were tried, the most pathological one being the Burstein's More Difficult switchbox, which we have shown in Fig. 6. For this switchbox, all the nets but one were successfully routed by the templates in the shortest possible path. Net number 3 was the only net that required use of the maze router.

Router	No. vias	Net-length	runtime(secs)
Mighty	39	541	4
Beaver	34	536	1
Mole	36	542	0.8

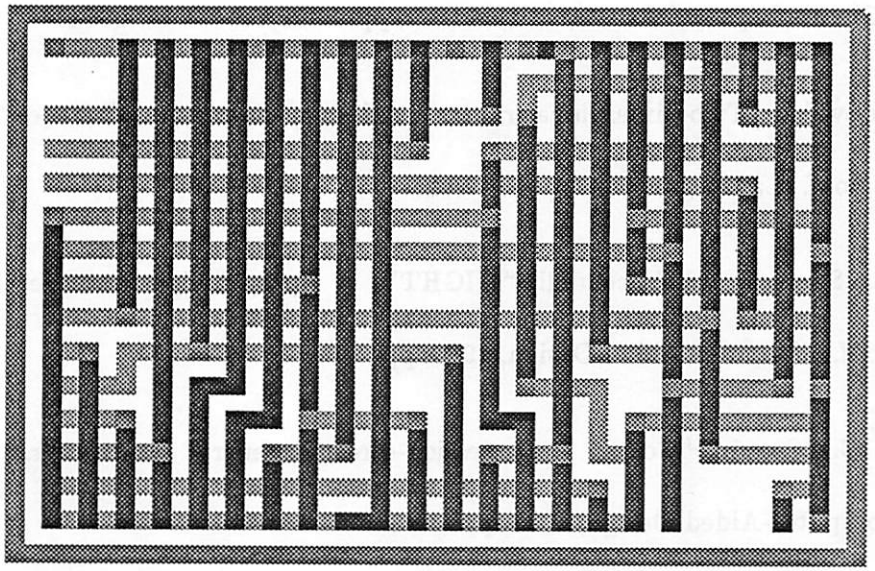


Figure 6: Routing result for Burstein's more Difficult Switchbox

References

[1] T.Asano, "Parametric Pattern Router," Proc. ACM/IEEE 19th Design Automation Conference, pp. 411-417, 1982.

[2] J.P.Cohoon and P.L.Heck, "BEAVER: A Computational-Geometry Based Tool for Switchbox Routing," IEEE Trans. Computer-Aided Design, Vol. 7, No. 6, pp. 684-697, June 1988.

[3] R.Joobbani and D.P.Siewiorek, "Weaver: A Knowledge based routing expert," IEEE Design & Test, Vol. 3, No. 1, pp. 12-33, Feb. 1986.

[4] J.Soukup, "Circuit Layout," Proc. IEEE, vol. 69, pp. 1281-1304, Oct. 1981.

[5] M.Marek-Sadowska, "Two-dimensional router for double layer layout," Proc. 22nd Design Automation Conf., Las Vegas, 1985, pp. 117-123.

[6] Y.Shin and A.Sangiovanni-Vincentelli, "MIGHTY: A rip-up and reroute detailed router," Proc. Int. Conf. Computer-Aided Design, 1986, pp. 2-5.

[7] P.Tzeng and C.H.Sequin, "Codar: A congestion-directed general area router," Proc. IEEE Int. Conf. Computer-Aided Design, 1988, pp. 30-33.

[8] T.M.Parng, E.S.Kuh and R.S.Tsay, "A unified approach to general-area routing," manuscript, 1989.

[9] R.S.Tsay, E.S.Kuh and C.P.Hsu, "PROUD: A sea-of-gates placement algorithm," IEEE Design & Test of Computers, pp. 44-56, Dec. 1988.