NUTCRACKER: AN INTELLIGENT CHANNEL SPACER

by

Xiao-Ming Xiong

Memorandum No. UCB/ERL M86/55

15 May 1986

NUTCRACKER: AN INTELLIGENT CHANNEL SPACER

by

Xiao-Ming Xiong

Memorandum No. UCB/ERL M86/55

15 May 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Nutcracker: An Intelligent Channel Spacer

Xiao-Ming Xiong

Electronics Research Laboratory
Department of EECS
University of California, Berkeley
Berkeley, California 94720

ABSTRACT

We present an intelligent channel spacer which can either transfer the symbolic channel routing result into physical layout or serve as a post-process step to compact the physically routed channel.

Most of the today's channel routers assume that the wires and contacts used to connect signal nets in the channel must be put on "tracks". The goal of the channel router is then to minimize the number of tracks in the channel. As long as the design rules are not violated, there is no reason to lay the wires and contacts on "tracks" in channel routing. Also the design rules for wires and contacts are usually different in today's fabrication technologies. Obviously, the height of channel can be reduced if we can make full use of these two advantages.

The goal of our channel spacer is to minimize the channel height by repositioning the wires and contacts according to the design rules given.

Nutcracker has two important features: to adjust the relative contact-wire position and to insert jogs when necessary.

Because the abuse of jogs will not only increase the total wire length in the channel but also make the physical layout very irregular, Nutcracker will insert jogs in an intelligent way so that a jog is inserted if and only if we can benefit.

The algorithm used for Nutcracker is very efficient in both aspects of complexities and actual run time. The worst case run time complexity is O(nlogn) and space complexity is O(n) where n is the number of wires and contacts in the final physical layout. Nutcracker is applicable to both one and two metal layer technologies and is suitable for both custom chips and gate arrays. Nutcracker could be a post-process step for many existing channel routers.

Nutcracker is implemented in C on a VAX 11/780 and used as a post-process step for the gridless channel router -- *glitter* ([2]). A lot of practical examples are tested and the experimental results shows that on the average the channel height can be reduced by about ten percent. The lower bound of the channel height in one dimensional compaction is achieved if the routing topology is given. Due to the inherent properties of the scan-line approach, the irregular channel boundary does not affect the performance of Nutcracker.

This research is sponsored by the Semiconductor Research Cooperation under SRC 82-11-008.

Nutcracker: An Intelligent Channel Spacer

Xiao-Ming Xiong

Electronics Research Laboratory
Department of EECS
University of California, Berkeley
Berkeley, California 94720

1. Introduction

Spacer is a CAD tool which is used to transfer rough sketches or symbolic diagrams to produce physical VLSI layout using design rules, or to minimize the chip area by compacting objects on a layout plane. If we can successfully compact physical layout, we can generate an initial physical layout from rough sketches or symbolic diagrams and then do the compaction. From this point on, we will concentrate on compaction. Compaction was used to describe the process of repeated cutting of unnecessary spaces from the given rough layout. Different methodologies and algorithms for layout compaction are developed and described in the literature ([1]). Most of the compactors developed so far are based on a one-dimensional approach using the longest path search in the horizontal and the vertical constrain graphs as used by Hsueh and Pederson ([3]). The constraint-graph approach offers a good flexibility and allows for an efficient implementation. However, this approach does not fit well with the compaction problem involving two dimensions. The boolean decision variables have been used by Schlag, Liao and Wong ([4]) in a way to formulate and solve the two-dimensional compaction problem. It has been proved that the simultaneous two-dimensional compaction problem is NP-complete ([4]).

An important problem in layout compaction is compacting routed channels ([5]). In channel routing compaction, we need only to consider two kinds of objects: wires and contacts. The goal of channel routing compaction is to minimize the channel height. Most of the today's channel routers assume that the wires and contacts used to connect signal nets in the channel must be put on "tracks". The goal of the channel router is then to minimize the number of tracks in the

channel. As long as the design rules are not violated, there is no reason to lay the wires and contacts on "tracks" in channel routing. Also the design rules for wires and contacts are usually different in today's fabrication technologies. Obviously, the height of a channel can be reduced if we can take full advantage of these characteristics.

We present an intelligent channel spacer -- Nutcracker, which can transfer the symbolic channel routing result into a physical layout or serve as a post-process step to compact the physically routed channel. The objective of Nutcracker is to minimize the channel height by repositioning the wires and contacts according to the design rules given. Nutcracker has two important features: to adjust the relative contact-wire position and to insert jogs when necessary. Because the abuse of jogs will not only increase the total wire length in the channel but also make the physical layout very irregular, Nutcracker will insert jogs in an intelligent way so that a jog is inserted if and only if an improvement is made. The final compaction result is dependent on design rules provided and the amount of channel height reduction is strongly dependent on the channel router used.

We only use the plane sweep technique in Nutcracker. Compared with other compaction algorithm which implement constraint graphs, our algorithm maintains and handles the data only on a scan line. Thus the algorithm used for Nutcracker is more efficient in both complexity and actual run time. The worst case run time complexity is O(nlogn) and space complexity is O(n) where n is the number of wires and contacts in the final physical layout. Another important feature of Nutcracker is that the performance of Nutcracker is independent of the shape of the channel boundary. This is because we use the scan-line approach to find all possible jogs in the first phase of the algorithm and the lower bound of the channel height is then achieved.

Nutcracker is applicable to both one and two metal layer technologies and is suitable for both custom chips and gate arrays. It can also be used as a post-process step for many existing channel routers.

2. The Scan Line Method

The scan line approach which has been widely described in the literature is efficient in handling geometric figures on a plane. For example, Preparata and Nievergelt [6] presented an implementation where the worst case running time, O(nlogn) -- n is the total number of corner points and intersecting points of the geometric figures on the plane, is the best currently known.

Let us give a brief overview of the scan line approach. For identifying and manipulating the geometric figures on a plane, we usually need to scan the plane at least once. To scan the plane in the vertical direction, a scan line is defined as a horizontal line which sweeps across the plane from bottom up. Similarly, we define scanning in the horizontal direction, say from left to right. When a object is hit by the horizontal scan line, we put it in an on-line list. When the object is ending at the scan line, we delete the line segment from the on-line list. Thus all objects on the plane will show up in the on-line list once. At any time, the on-line list contains all the geometric figures which intersect the scan line and these are the geometric figures which can be maintained and manipulated.

The scan line approach is very efficient since it actually transforms the two-dimensional problem into a one dimensional problem. The key aspect of the scan line approach is to be able to maintain the necessary information in the on-line list.

3. Glossary of Terms

Net: A net is defined by a set of terminals that must be interconnected by some routing path.

Channel: A channel is a rectangular routing region with terminals along its edges. There are two kinds of channels: horizontal channel and vertical channel. In a horizontal channel, terminals are along the top and bottom edges of the channel and we can shrink or expand the channel in the vertical direction. In a vertical channel, terminals are along the left and right edges of the channel and we can shrink or expand the channel in the horizontal direction. Without loss of generality, we will assume that the channel

under consideration is a horizontal channel.

Wire: A wire is a straight line segment which is a portion of a routing path. There are two types of wires: horizontal wires and vertical wires. To compact a horizontal channel, we do not care about the vertical wires in the channel.

Contact: A contact is a feed-through used to connect two wires on different layers in the same routing path.

Jog: A jog is a vertical wire which connects two horizontal wires of the same routing path on the same layer.

4. Underlying Idea

In most of the existing compaction algorithms, a constraint graph is first build by sweeping the layout plane from left to right. Then apply the longest path algorithm to the constraint graph to do the compaction. In contrast to other algorithms, ours does not use constraint graph and the compaction is done by sweeping the channel from bottom up.

We call both the horizontal wires and contacts in the channel *objects*. An object can be represented by its center line associated with a number, hw, which is half of the object width. *A discontinuous point* is either the point shared by two adjacent objects or the end point of an object which has no adjacent objects. Contacts C1, C2, C3 and wires W1, W2 in Fig. 1 is respectively represented as objects objc1, objc2, objc3, objw1, and objw2 in Fig. 2. The points A, B, C, D, E, and F in Fig. 2 are discontinuous points. The half width, hw, is two for objc1, objc2, and objc3 but hw equals one for objw1 and objw2.

An object *covers* a discontinuous point if the object is right above the point and the the x-coordinate of the point is in between of the x-coordinates of two end points of the object. In Fig. 2, the object objw2 covers the discontinuous points A, B, and C; objc3 covers discontinuous points C and D. Notice that a discontinuous point can be covered by more than one object.

Our algorithm consists of two phases. In the first phase, we find all possible jog positions and assign a weight to every object by scan the channel from bottom up. Notice that all possible jog positions are the discontinuous points but a discontinuous point is not necessary a possible position for a jog, e. g. A and D in Fig. 2 are not possible jog positions. The channel height is determined in this phase. Then in the second phase, we embed wires and contacts in the channel from top down and minimize the number of jogs under channel height obtained in the first phase.

We sort all objects in the channel lexicographically by their y, x coordinates and then sweep the channel by jumping a scan line at the y coordinate of the objects. By *current scan line*, we mean the position where the scan line is currently at. By *previous scan line*, we mean the position where the scan line is just jumped from. A linked list, P, is used to maintain all discontinuous points on the previous scan line. For every object hit by current scan line we do the following. If the object covers some discontinuous points in P and the object is longer enough to insert a jog, then we will break the object into two objects at the possible jog position and generate a new discontinuous point. Otherwise, consider the next object. All discontinuous points covered by certain object will be deleted from P and all discontinuous points on current scan line will be inserted in P as scan line jumps. For example, objw2 in Fig. 2 is broken into four objects objw21, objw22, objw23, and objw24 and new discontinuous points N1, N2, and N3 are generated (see Fig. 3). The object, objc3, can not be broken because there is no room for a jog (Fig. 3). An object A is said to be covered by another object B if B is right above A and when A moves up at the same y-coordinate as B, A and B intersect each other. The weight of the bottom boundary edge of the channel is its y coordinate and a weight, W_o, is assigned to an object, o, by the following formula while we are scanning the channel.

$$W_o = \max (W_i + hw_i) + hw_o + vc$$

where max means take the maximum over all i's; W_i is the weight of object i covered by o; hw_i is the half width of object i; hw_o is the half width of object o; vc is the vertical clearance between two objects given by design rule. Notice that the weight of an object is actually the smallest value of y coordinate this object can have without violating the given design rules. The weight of

the top boundary edge of the channel determines the minimal channel height.

After scanning the channel, we sort all objects by their weights in a linked list, L. We embed the object with maximal weight in L by changing its y coordinate to its weight and then embed the objects adjacent to the object we just embedded. If design rules are not violated, we place the adjacent objects at the same y-coordinate. Otherwise, change the y coordinates of the adjacent objects to their weights and jogs are added to connect two adjacent objects with different y-coordinates. Deleted embedded objects from L and repeat the above procedure until all objects are embedded.

Fig. 4 gives an example of a routed channel. Fig. 5 shows the result after compaction.

5. Pseudo Code of the Algorithm

Nutcracker(R)

INPUT: Channel routing output R = (B, W, C), where B is the edges of the channel boundary; W and C are respectively a set of wires and a set of contacts in the channel.

OUTPUT: Updated R: the physical layout of the channel with minimal height.

METHOD: scan line approach

```
begin

    J = FindAllPossibleJogs( R );
    MinimizeJogs( J, R );

end
```

FindAllPossibleJogs(R)

INPUT: Channel routing output R = (B, W, C), where B is the edges of the channel boundary; W and C are respectively a set of wires and a set of contacts in the channel.

OUTPUT: J: a set of possible jogs.

METHOD: Sweep chip plane from bottom up to identify all possible jog positions; at the mean time we will calculate the weight of every object which is the smallest y-coordinate where the object can be placed at without violating the design rules.

```
begin

    objects = list of objects sorted lexicographically by their y and x coordinates;
    previous_scan_line = y-coordinate of bottom boundary edge of the channel;
    current_scan_line = y-coordinate of first object in objects;
    while( current_scan_line <= y-coordinate of top boundary edge of the channel ) {
        while( y-coordinate of object == current_scan_line ) {
            put objects in list on current_scan_line;
            if( an object covers some discontinuous point in previous_scan_line &&
            there is room for a jog )
                break the object and add a new discontinuous point in
                current_scan_line;
            put all discontinuous points not covered by the objects on
            current_scan_line from previous_scan_line to current_scan_line;
            calculate the weights for all objects on current_scan_line;
        }
        previous_scan_line = current_scan_line;
        current_scan_line = y-coordinate of the object;
    }
    generate possible jogs;
    output all possible jogs;

end
```

MinimizeJogs(J, R);

INPUT: Channel routing output R = (B, W, C), where B is the edges of the channel boundary; W and C are respectively a set of wires and a set of contacts in the channel. J is a set of all possible jogs in the channel.

OUTPUT: Updated R: the physical layout of the channel with minimal height.

METHOD: Embedding all objects in the channel from top down. The object with maximal weight is embedded first and then its adjacent objects are considered to be put in the position where no jog will be introduce. The goal of this step is to embed objects in the channel and minimize the number of jogs under fixed channel height.

```
begin

    objects = list of objects sorted by their weight;
    object = first object in the list of objects;
    while( object != NULL ) {
        if( object not marked ) {
            y-coordinate of objects = weight of the object;
            while( object has adjacent object ) {
                if( adjacent object can be placed at same y-coordinate of object without
                violating design rule ) {
                    y-coordinate of adjacent object = y-coordinate of object;
                    delete the possible jog associated with the object in J;
                }
                else
                    y-coordinate of adjacent object = maximal y-coordinate it can be
                    placed without violating design rule;
                mark this adjacent object;
            }
            mark object;
        }
        object = next object in the list;
    }
    generate jogs according to J;
    output R;

end
```

6. Complexity Analysis

Because we simply use the plane sweep algorithm to do the compaction, the complexities of our algorithm are the same as the plane sweep algorithm. Let n be total number of objects in the channel. The worst case time complexity of the algorithm is O(nlogn) if a balanced tree is used to maintain the data on a scan line. If a linked list is used to maintain the data on a scan line, the worst case time complexity of the algorithm will be $O(n^2)$. If n is large and all objects are uniformly distributed in the channel (this is true in channel routing problem), then the average time complexity for two cases are both linear because we have approximately $n^{0.5}$ scan-lines and each scan line will intersect about $n^{0.5}$ objects. Due to the constant in time complexity, in actual run time, the linked list implementation of the algorithm is much faster than the balanced tree implementation of the algorithm. Our experimental results verify the above statements. The space complexity for the algorithm is O(n).

7. Experimental Results

Nutcracker is implemented in C on a VAX 11/780 and used is as a post-process step for the gridless channel router – *glitter* ([2]). A lot of practical examples have been tested, and experimental results (see table 1) indicate that on the average, the channel height is reduced by about ten percent. All examples in table 1 were finished in less than 3 seconds on a VAX/780. According to our experimental results, the lower bound of the channel height in one dimensional compaction is always achieved.

8. Concluding Remarks

We conclude with some further discussions on our channel spacer. If the routing topology is fixed, the lower bound of the channel height is achieved by Nutcracker. According to our experimental results, the channel height can be reduced by about ten percent using Nutcracker as a post-process step. The algorithm used in Nutcracker is very efficient in terms of both complexity and actual run time. Currently, we are enhancing Nutcracker to handle channels with irregular

boundaries and to allow contacts moving in both directions. We are also trying to extend the approach used in Nutcracker to attack more general compaction problems.

9. Acknowledgment

The author is very grateful to his research advisor Professor E. S. Kuh for his constant support and encouragement. It has been a most rewarding experience to work under his guidance. I would also like to thank Dr. U. Lauther, Dr. M. Sato, H. H. Chen, W. M. Dai and M. Jackson for their helpful discussions and valuable suggestions during the development of this work.

Table 1

Experimental Result
of Routed Channel Compaction

channel	number of nets	number of pins	router height	spacer height	CPU time(s)	gain %
small	10	22	37	34	0.05	8.1
amd	8	16	322	194	0.04	39.75
Average						23.9
hughes1	221	489	1790	1550	2.4	13.4
hughes2	252	560	1470	1270	2.6	13.6
hughes3	234	499	1310	1050	2.5	19.8
hughes4	230	502	2010	1810	2.7	9.95
hughes5	85	239	1250	1210	2.0	3.2
hughes6	139	359	1570	1330	2.7	15.28
hughes7	133	343	1410	1330	2.4	5.67
hughes8	133	318	1570	1450	2.5	7.64
hughes9	117	288	930	890	1.4	4.3
hughes10	129	319	1090	1010	1.8	7.34
hughes11	113	293	1170	1110	2.4	5.13
hughes12	121	284	1170	1070	1.9	8.55
Average						9.46

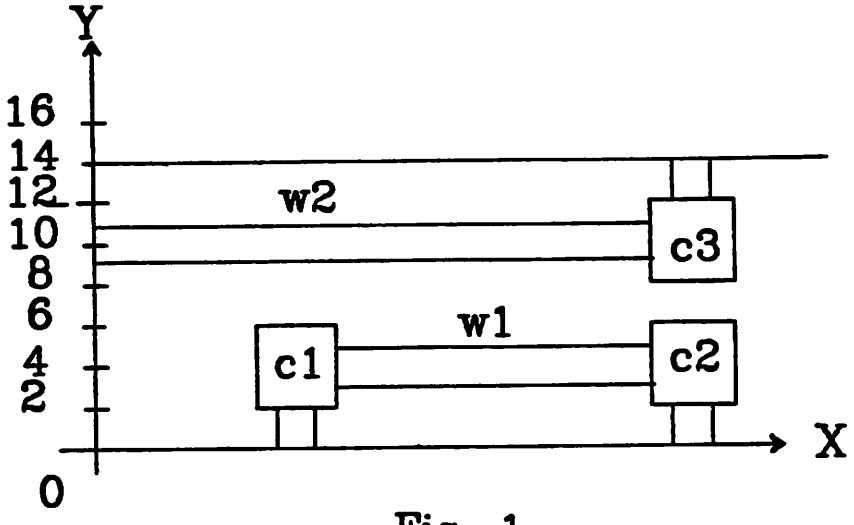

Fig. 1

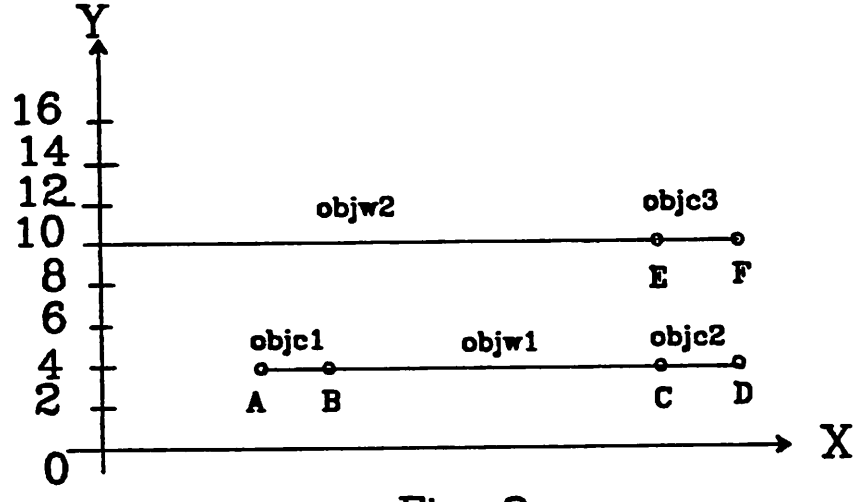

Fig. 2

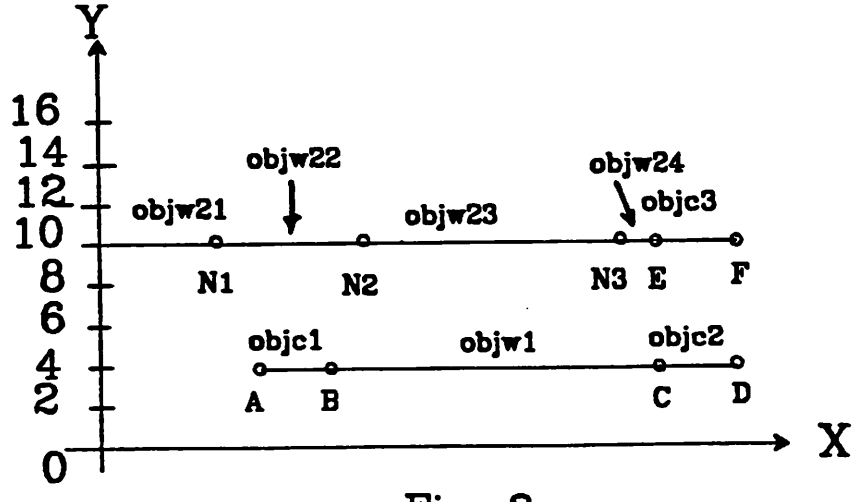

Fig. 3

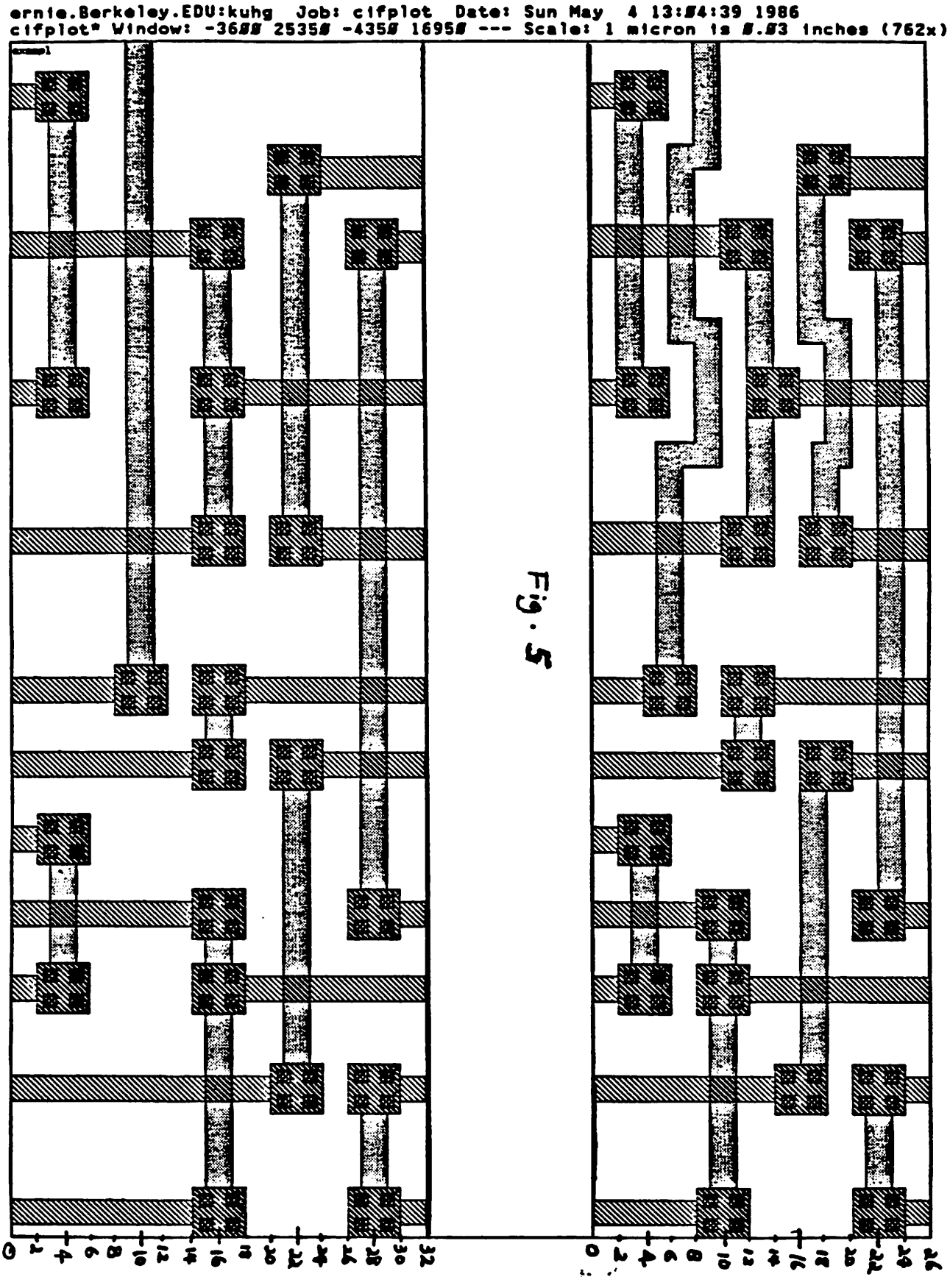
ernie.Berkeley.EDU:kuhg Job: cifplot Date: Sun May 4 13:#4:39 1986
cifplot* Window: -36## 2535# -435# 1695# --- Scale: 1 micron is #.#3 inches (762x)
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32

Fig. 4

Fig. 5

References

[1]. D. D. Mlynski and C. H. Sung, "Layout Compaction", Advances in CAD for VLSI, Vol. 4, T. Ohtsuki Editor, North Holland Publ. Co., 1985.

[2]. H. H. Chen and E. S. Kuh, "A Variable-Width Gridless Channel Router", Proc. of ICCAD-85, November 1985, pp. 304-306.

[3]. M. Y. Hsueh and D. O. Pederson, "Computer-Aided Layout of LSI Circuit Building Blocks", Proc. IEEE International Symposium on Circuits and Systems, 1979, pp. 474-477.

[4]. M. Schlag, Y. Z. Liao and C. K. Wong, "An Algorithm for Optimal Two-Dimensional Compaction of VLSI Layouts", Integration, VLSI Journal, Vol. 1, 1983, pp. 179-209.

[5]. D. N. Deutsch, "Compacted Channel Routing", Proc. of ICCAD-85, November 1985, pp. 223-225.

[6]. J. Nievergelt and F. P. Preparata, "Plane-Sweep Algorithms for Intersecting Geometric Figures", Communications of the ACM, Volume 25, Number 10, October 1982, pp. 739-747.