He received the Ph.D. in Electrical Engineering from Carnegie-Mellon University, Pittsburgh, PA, in 1982. In 1975 he received the first prize in the Copernicus Competition for the best student in the University. Since October 1976 he has been a member of the faculty of the Technical University of Warsaw. During the Summer 1981 he was working at HARRIS Semiconductor Co. on modeling and diagnosis of the IC manufacturing process. He is currently Assistant Professor of Electrical and Computer Engineering at Carnegie-Mellon University. His research interests include statistical modeling of IC fabircation processes and semiconductor devices, computer-aided design of VLSI circuits, and optimal control and diagnosis of the IC fabrication process.

*

Stephen W. Director (S'65-M'69-SM'75-F'78) received the B.S. degree from the State University of New York at Stony Brook, Stony Brook, NY, in 1965 and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, in 1967 and 1968, respectively.

He is Whitaker Professor and Head of the Department of Electrical and Computer Engineering, Professor of Computer Science and Director of the Research Center for Computer-Aided Design at Carngie-Mellon University. In 1968 he joined the Department of Electrical Engineering of the University of Florida, Gainesville, and was promoted to Professor in 1974. From September, 1974 to August, 1975 he was a Visiting Scientist in the Mathematicsl Sciences Department at IBM's T. J. Watson Research Center. Yorktown Heights, NY. He joined Carnegie-Mellon University as a Professor in 1977. His research interests are in the area of computer-aided design and numerical methods and he is a consultant to industry in this area. He is also a Consulting Editor to McGraw-Hill book Company.

Dr. Director is a member of Sigma XI and Eta Kappa Nu. In 1981 he served as President of the IEEE Circuits and Systems Society and has also served as Secretary-Treasurer, President-elect, and a member of the Administrative Committee of this Society. Further, he has been Chairman of the CAS Technical Committee on Computer-Aided Network Design (CANDE) and served as an associate editor of the IEEE Transactions on Circuits and Systems, as well as a guest editor of this Transactions and the Proceedings of the IEEE for special issues on computer-aided design. In addition to writing numerous technical articles, he has authored or coauthored three texts.

In 1970 he received the Best Paper Award from the IEEE Circuits and Systems Society; in 1976 he received the Frederick Emmons Terman Award as the outstanding young electrical engineering educator from the American Society of Engineering Education; and in 1978 he received the W.R.G. Baker Prize Paper Award from the IEEE.

A Procedure for Placement of Standard-Cell VLSI Circuits

ALFRED E. DUNLOP, MEMBER, IEEE, AND BRIAN W. KERNIGHAN, MEMBER, IEEE

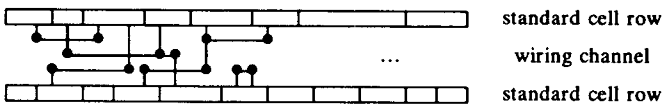
***Abstract*—This paper describes a method of automatic placement for standard cells (polycells) that yields areas within 10–20 percent of careful hand placements. The method is based on graph partitioning to identify groups of modules that ought to be close to each other, and a technique for properly accounting for external connections at each level of partitioning. The placement procedure is in production use as part of an automated design system; it has been used in the design of more than 40 chips, in CMOS, NMOS, and bipolar technologies.**

I. Introduction

A MAJOR problem in VLSI is: given the logical design of a circuit, position the components and route the connections between them to minimize the area required. Sometimes this is done to minimize the number of chips necessary to realize a function; more often it is to get as many instances of a chip as possible on a wafer.

Manuscript received November 28, 1983; revised July 31, 1984.
The authors are with AT&T Bell Laboratories, Murray Hill, NJ 07974.

The specific variant of this problem that we are considering arises when the components of the circuit are "standard cells" or "polycells." Standard cells are typically designed so as to fit together in rows. Common signals like power, ground and possibly clocks run through the cells horizontally at fixed positions. Each cell has terminals on its top and/or bottom; the circuit is wired by connecting these terminals with wires that run in channels between and around the rows of cells. Feedthroughs may be used to make connections that span more than one row. Horizontal and vertical connections are on distinct layers so there is no electrical connection at a crossing unless one is explicitly created. These notions are illustrated in the following figure:



As an aside, it is common to use standard cells for much of the logic of a circuit, but also to use larger building blocks like PLA's, memories and register banks.

Standard cells have many advantages as a design style; in particular, design time can be quite short and layout is straightforward if there is enough area. Unfortunately, however, in most circuits, a high fraction of the area (50–70 percent is not uncommon) is devoted to the wiring between the cells. Thus it is important to develop procedures that will perform efficient placement to minimize routing area. In this paper, we will be concerned only with placement; routing is left to some other program, although our placement is intended to make routing in a small area an easier job.

One class of placement heuristics might be called "constructive." One starts with one or a few "seed" cells and adds others that seem to be closely related to what has gone before. The intent is to keep objects that are connected close to each other so that the wires between them are short and thus require as little area as possible.

A second broad class is "iterative improvement." An arbitrary placement is created without regard to quality; it is then improved in stages, often by rearranging small groups of cells, until no further improvement can be found. This process can be repeated as many times as desired; each new initial placement leads (at least potentially) to a different and perhaps better final placement.

Of course it is possible to use these two strategies together, most often by using an iterative technique to polish a carefully constructed original placement. Our method is such a combination. It uses graph partitioning to identify clusters, i.e., groups of cells that are closely related. The idea is that cells within a cluster ought to appear close to each other in the final placement, so as to waste little area routing among them. Clusters are then assigned to adjacent rows, preserving nearness of related cells. Finally, other tools (not part of the discussion in this paper) are used to polish the solution: an iterative cell-exchange program such as PRO [1] is used to improve each row in isolation; a channel router like Deutsch's [2] does the final routing within the channels.

Identifying clusters is intuitively appealing, and various forms of this attack have been reported in the literature for many years, under names like partitioning, nested bisection, and min-cut placement [3]-[7].

One distinguishing feature that appears to contribute markedly to the success of our method is a novel way to do "terminal propagation," that is, to use information like the positions of external connectors and estimates of the positions of unplaced cells to guide subsequent partitioning. Our terminal propagation uses Steiner trees to approximate the effect of external connections. Other algorithms have been based on a force analogy [8] or routing length [9].

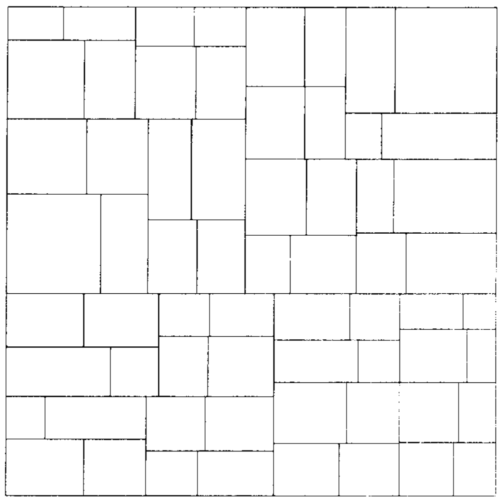
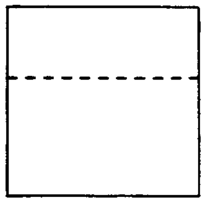


Fig. 1. Partitioning of 412-cell circuit into groups of size ≤ 8.

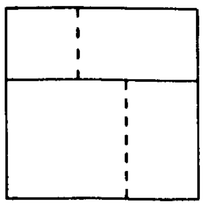
II. The Procedure

We begin with the logic description of a circuit (to which cells must be electrically connected), but no geometry. The idea is that the cells are partitioned first into two groups, then into four, then into eight, and so on, until there are only a few elements in each group. Suppose that horizontal rows of cells are desired. Then the first step of partitioning divides the n cells into two sets of $n/2$ cells like this:



The number of cells in each side is the same; the area of each half is proportional to the area of the cells included in it.

The next step of partitioning is to divide each of these groups in two, this time on the other axis:



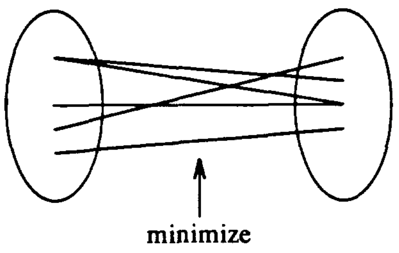
This process continues until there are only a few cells in each group (typically half a dozen). The effect is that at each level cells are localized to the region in which they ought to be finally located, but their actual placement is not fixed. Fig. 1 shows a partitioning of a circuit of 412 cells into 64 groups of size ≤ 8.

At this point, it is possible to assign the cells in each group to rows, keeping those cells that ought to be together (as determined by the partitioning) either close together in the same row or nearby in an adjacent row.

But layouts obtained merely by partitioning cells and assigning them to rows are not nearly as good as they might be. A second major component of the procedure, then, is to take into account any positioning information that may be present, such as the locations of external pin connections and the probable locations of cells in the final placement. This is called *terminal propagation;* it is done concurrently with partitioning.

Partitioning

The original graph partitioning problem is: given n nodes connected by edges (possibly with weights greater than 1), partition the nodes into two subsets of size $n/2$ such that the sum of the weights on the edges connecting the two subsets is minimum:



The graph partitioning problem for circuits is very similar. The nodes are circuit elements, and the edges become *nets* that connect two or more elements. A net is said to be *cut* if it connects elements in one subset to elements in the other. The cost of a partition is the number of nets that are cut.[10]

Graph partitioning, in either form, is NP-complete [11] but there are good heuristics, such as the one developed by Kernighan and Lin [12], [13]. The strategy is iterative improvement. Given an arbitrary initial partition into two sets A and B, there is some number of elements of A and an equal number of elements of B that are "out of place," in the sense that if they were interchanged, the resulting partition would be optimum. The Kernighan-Lin procedure tries to identify these out-of-place elements sequentially.

Let $a \in A$ and $b \in B$, and let N be a net. If a is the only element of N in A, then moving a to B reduces the number of nets cut by 1. Let E_a be the number of nets touching a for which a is the only element in A.

Similarly, if all elements of a net are in one set, say A, then moving one element of the net to B causes the net to be cut and thus costs 1. Let I_a be the number of nets touching a which have all their elements in A.

Clearly, I_a measures how strongly a is connected to A, while E_a measures how strongly it is attracted to B. Then $D_a = E_a - I_a$ is a measure of how out of place a is: the larger D_a is, the more likely that a belongs in B instead of A. We define D_b similarly.

The gain g obtained by exchanging a and b is simply

$$g = D_a + D_b - c_{ab}$$

where c_{ab} is a correction that avoids double counting if a and b are both on some net.

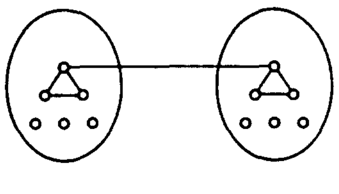
The procedure is as follows:

```
repeat {
    $A' = A; B' = B$
    for $i = 1$ to $n/2$ {
        compute $D$ values for all $a \in A'$ and $b \in B'$
        find $a_i$ and $b_i$ that maximize $g_i = D_{a_i} + D_{b_i} - c_{a_i b_i}$
        move $a_i$ to $B$ and $b_i$ to $A$
        remove $a_i$ and $b_i$ from further consideration this pass
    }
    find $k$ that maximizes $g_{\max} = \Sigma_{i=1}^{k} g_i$
    if $g_{\max} > 0$
        exchange $a_1, \cdots, a_k$ with $b_1, \cdots, b_k$
} until $g_{\max} = 0$
```

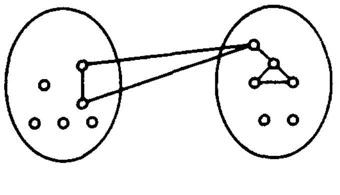
Each time a pair a and b is selected, they contribute to the gain so far, which is just Σg_i. Notice that once a pair has been selected, the elements are set aside and are not eligible to be selected again on this cycle. The value of k, $1 \leqslant k \leqslant n/2$, which makes the sum of gains as large as possible determines whether an improvement has been made. If there is a gain, $g_{\max}$ will be positive; in that case, $a_1, \cdots, a_k$ are exchanged with $b_1, \cdots, b_k$ and the process is attempted again. If the improvement $g_{\max}$ is zero, the resulting partition is a local minimum. Typically this takes 2-5 iterations of the repeat loop for partitions of a few hundred objects.

The partial sum Σg_i may actually be negative in the early stages, yet later become positive, for instance, during the exchange of a cluster. Moving the first few elements of a cluster raises the cost, but it falls again when the entire cluster is moved. Thus the algorithm is able to escape from some local minima that would trap a simpler procedure.

To illustrate, consider the situation below, where each edge is a separate net. The cost is 1:



If a single element on some net is moved from left to right or vice versa, the cost rises to 2, so the "gain" g_1 is actually -1:



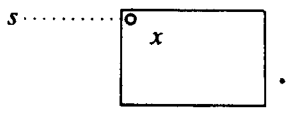
Moving one more connected element from left to right leaves the cost unchanged at 2 (i.e., $g_1 + g_2 = -1$). Only after all three elements have moved does the cost become zero, and thus the total gain $g_{\max}$ becomes 1.

Computation of the D values can be done in time proportional to the number of pins in the circuit. For most circuits, the number of pins per element is small. Thus computation of D values requires $O(n)$ time for n elements, and time $O(n^2)$ to do it n times. Finding the maximum a_i and b_i requires $O(n \log n)$ for each pass, and thus $O(n^2 \log n)$ overall. Computing the correction in the worst case could take $O(n^2)$ but in practice it is much smaller. The overall time complexity of the partitioning procedure is thus around $O(n^2 \log n)$ as we have implemented it.

In circuit problems, it is generally assumed that all nets have unit cost, but it is possible to create multiple instances of or apply a weight (greater than one) to a net and thus prevent (or at least bias) it from being split during partitioning. It is also possible to obtain partitions which vary from exactly $n/2$ elements in each subset by adding dummy elements to the original set. These dummy elements are not connected to anything, so they have no effect on the cost, but they permit an unbalanced partition of the real elements.

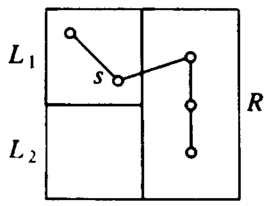
Terminal Propagation

It is not adequate merely to partition groups of the circuit in isolation, since the signals that enter a group of cells from the outside effect where the cells ought to be placed just as much as the internal connections among the group. We call the procedure for taking this into account "terminal propagation." As the simplest example, consider a group of cells containing a cell x that is connected to a signal s from outside the group, for example, from a pad:



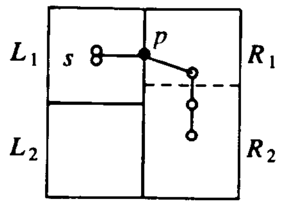
All other things being equal, cell x ought to be near the point where signal s comes from.

This is clearest at the outermost level, where the signal positions are typically fixed by pad positions or the positions of other components of the circuit. But what happens at a lower level during partitioning? Suppose we have partitioned the cells into groups L and R, then partitioned L into L_1 and L_2, and that there is a signal net s that connects two cells in L_1 with three in R:



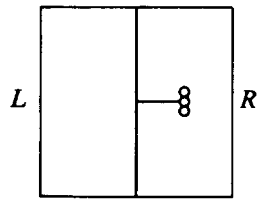
If we now want to partition R into R_1 and R_2, we would like to take into account the fact that signal s is in L_1 but not in L_2 and thus bias the partitioning process towards putting the cells into R_1 instead of R_2.

This is done as follows. In lieu of any other information, we assume that all of the cells of signal s in L_1 are at its geometrical center, and propagate that position to the closes point, say p, on R:



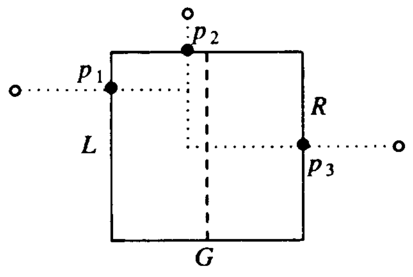
During partitioning, signal s is required to remain in the set R_1. This biases the partitioning process: the cost of partitioning R will be 1 lower if signal s appears only in R_1 than if some or all cells containing s are in R_2. Without this bias, there would be nothing to favor R_1 over R_2.

This example conveys the essence of terminal propagation, but there are many more situations to be considered. For example, assume that we are at the previous stage of partitioning, where neither L nor R has yet been partitioned. When we are partitioning L, the three elements of s in R are assumed to be concentrated at the middle of R, which propagates to the middle of L:



External signals that propagate to a point near the axis about which partitioning is to be done should not be used to bias the solution in either direction; accordingly, the elements of s in R should have no effect on how L is partitioned. The measure of "near" is arbitrarily set at "within the middle third of the side."

In the general case, suppose we are partitioning a group G into L and R, and that some net s has elements both inside and outside of G. (The "elements" may be individual cells or sets of cells at the centers of other groups.) We compute a low-cost rectilinear Steiner tree[1] on the elements external to G, and find the points of intersection $\{p_i\}$ with the border of G, as sketched below:



These points are treated as cells on s, but are fixed during partitioning, so they can not move from whichever of L or R they

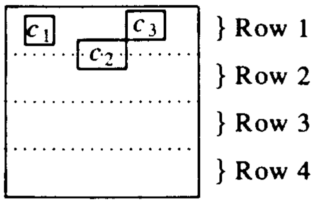
[1]Finding a minimum-cost rectilinear Steiner tree is another NP-complete problem, for which we use a heuristic based on a procedure described by Hanan [14].

are in. As before, we exclude the middle third of the sides perpendicular to the axis of partitioning, so any p_i that fall in these regions are simply ignored. Thus in the situation illustrated above, p_2 would be ignored, since it is within the middle third of the top side. But if partitioning were being done about the other axis, p_3 would be ignored and signal s would be biased to the top half.

The terminal propagation step makes a substantial difference in the quality of solutions obtained, reducing the overall area by about 30 percent. Notice that the use of terminal propagation implies that the partitioning must be done breadth-first, not depth-first: there is no point in working hard to partition one side to finer and finer levels when one has only the coarsest idea of the positioning on the other side.

Creating Rows

The final step is to create rows from the final partition (i.e., at the lowest level). When partitioning reaches the bottom level, we have identified for each cell the approximate region that it ought to be in, and its neighbors. Suppose for example that there are to be four rows in the horizontal direction. Then conceptually the layout area is divided into four strips like this:



Cells within the partition groups labeled c_1 and c_3 which lie entirely within the first strip are simply assigned to row 1; cells from c_1 would of course lie to the left of those from c_3.

Cells within a partition group like c_2 that lies across the boundary between one strip and the next are assigned to the two rows in question in proportion to the fraction of the partition group's area that lies in each row. In the figure above, for instance, c_2 is evenly divided between strips 1 and 2, so the cells within it are shared evenly (in area) between the rows. Because c_2 lies between c_1 and c_3, however, those cells from c_2 that go into row 1 lie between cells from c_1 and c_3.

This process keeps related cells together, and tends to keep the lengths of the rows much the same. Some minor rearrangement may be needed, however, to ensure that the row lengths are close enough to equal. Candidates for such rearrangements come from partition groups like c_2 that fall across a row boundary, since the choice of row for cells within c_2 is arbitrary.

Figs. 2 and 3 illustrate this process on a larger example.

III. Experience

The procedure is in production use as part of an automatic placement system. It has worked well on a variety of placement problems.

For one CMOS chip with 453 signals and 412 cells, a carefully tuned hand layout by one of the best layout engineers has a track density of 147 and 184 feedthroughs. (We use track density in this section so as to make comparisons independently of the

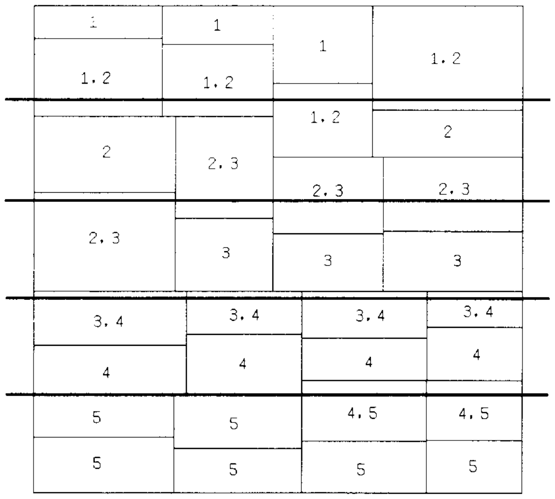


Fig. 2. Partitioning of circuit into 32 groups. Each group is either assigned to a single row or divided between two rows.

design rules and size of standard cells.) Without terminal propagation, the partitioning procedure achieved a track density of 313 (591 feedthroughs), which was subsequently reduced to 235 by iterative interchange. After terminal propagation was added, these numbers were substantially lower: track density 186 (182 feedthroughs), subsequently reduced to 152 by an automatic iterative interchange routine. The track count of 152 is within three percent of hand layout. Fig. 4 shows the final automatic placement and routing of this circuit.

This chip was used to investigate variations in the basic procedure; some results are shown in the table and graphs below.

Minimum Size	Variation Percent	Track Density	Feedthroughs	CPU Time (Seconds)
1	0	186	182	3230
1	5	199	174	3932
1	10	187	204	4278
2	0	186	178	3465
2	5	183	175	3702
2	10	196	202	4067
4	0	185	183	3271
4	5	187	172	3464
4	10	190	213	3842
8	0	197	215	3072
8	5	211	205	3275
8	10	200	227	3641
16	0	202	216	2911
16	5	195	211	3079
16	10	203	218	3414

These are based on the best partition found in 64 different starting arrangements at each level of partitioning. The starting point for the first iteration was obtained by dividing the cells in the original circuit description into two groups by their ordinal position in the logic description; this is often a good start, since the logic description tends to group related cells. Subsequent iterations began with a random arrangement of the cells. "Minimum size" indicates the size of group at which partitioning ceased; for example, a size of 8 indicates that no group of fewer than 8 cells would be partitioned. The "variation per-

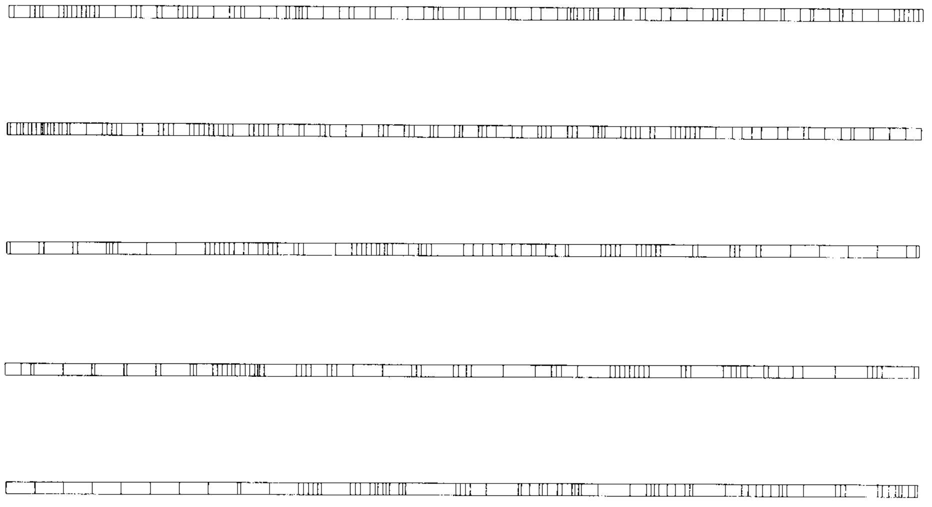


Fig. 3. The row assignment corresponding to Fig. 2.

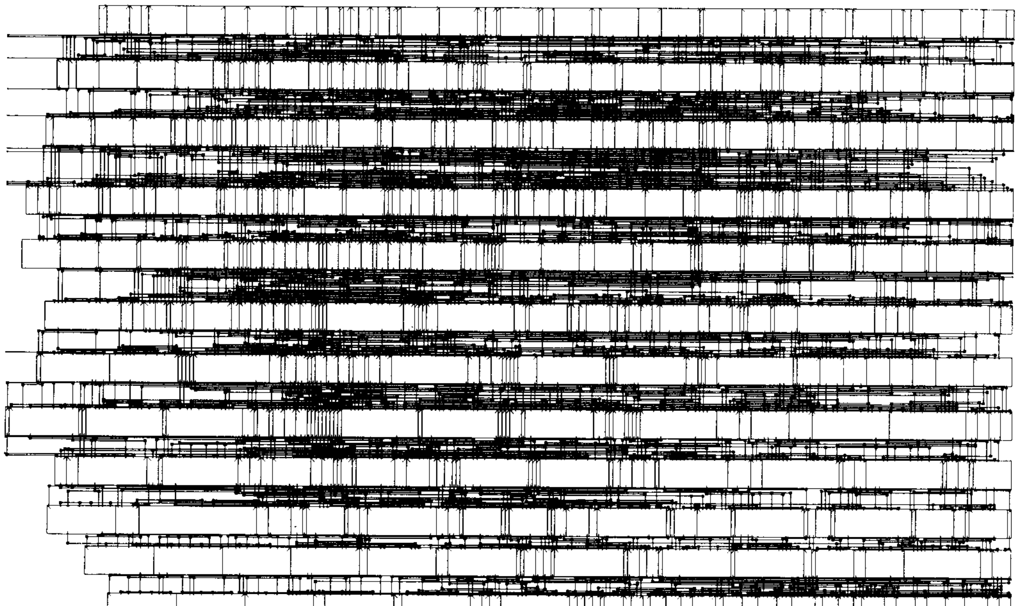
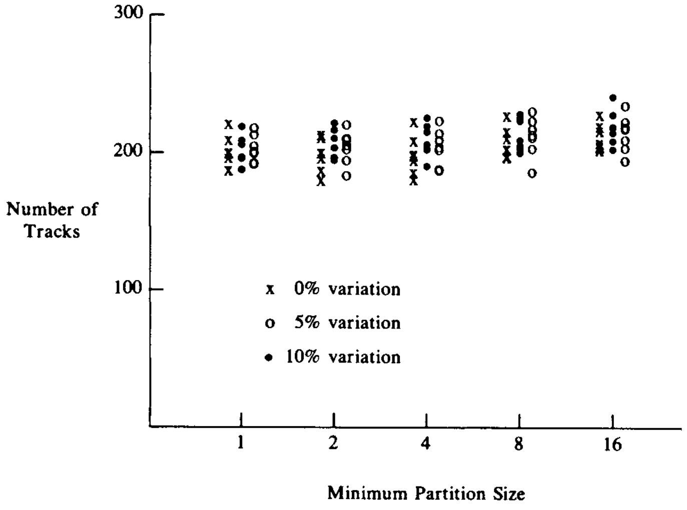


Fig. 4. Automatically generated layout for circuit with 453 signals and 412 cells.

cent" indicates the percent of dummy cells to be distributed between the two halves. This allows a variation in the number of cells that end up in the resulting two groups. The "feedthrough" column indicates how many feedthroughs were necessary to actually complete the routing. (We forced all nets to use feedthroughs instead of going around the ends of the cell rows.) Track density alone is not an adequate measure, since in a practical routing situation, feedthroughs have to be added either at the ends of rows or between cells within the rows to get signals between rows that are not adjacent; these add to the area. Finally, CPU times are for partitioning, terminal propagation, and conversion to rows, using a C program on a DEC VAX 11/780.

The graph on the right shows the same data, with other experiments done with 1, 2, 4, 8, 16, and 32 iterations as well. In virtually all cases, the more iterations, the better the results.



The following table summarizes experience with this and 3 other chips.

	Manual		Automatic		
Cells	Tracks	Fthru	Tracks	Fthru	Tech.
290	91	0	93	51	CMOS
412	147	184	152	182	CMOS
1158	444	268	484	501	NMOS
1198	581	410	384	441	CMOS

The automatically generated solutions are generally within 10 percent of the best hand layout.

IV. Extensions and Future Work

Recent work by Fiduccia and Mattheyses [15] shows that partitioning time can be substantially reduced. Their modification of the original Kernighan-Lin method requires time $O(P)$, where P is the total number of pins in the circuit. We have implemented a version of the Fiduccia-Mattheyses algorithm. It produces solutions much more quickly; it is about an order of magnitude faster for partitioning circuits of 2000 elements. It appears that the solution quality is not as good, however; track density is close to that of the traditional Kernighan-Lin method (but is sometimes as much as 15 percent larger).

We have also done some comparisons with a partitioning algorithm based on simulated annealing [16]. This technique seems to be substantially slower than the other procedures, but yields solutions of about the same quality as the Fiduccia-Mattheyses algorithm if we limit the CPU time to a reasonable value. It does have the advantage of being very simple to implement.

As mentioned above, the starting point for the first iteration was based on the original logic description of the circuit, and subsequent starts were completely random. We have since done some experiments in which each start is created by performing only a limited rearrangement of the best solution previously seen. This has two benefits: it produces better solutions, and it produces them faster, since fewer iterations are needed to go from initial partition to final. We intend to do more experimentation with this tactic.

We have also implemented a mechanism for forcing modules to be in particular positions ("seeding" or "hard placement"). We are still seeking an elegant solution to this problem; in the meantime, our approach is brute force.

The Steiner tree algorithm associated with terminal propagation is not optimum. We also plan to investigate other algorithms for terminal propagation; since partitioning is now so much faster, terminal propagation has become the bottleneck.

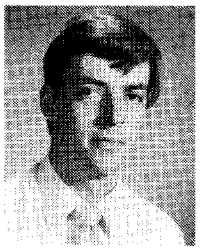
Acknowledgment

The authors are grateful to Dave Deutsch, Hermann Gummel, Steve Johnson, Linda Kaufman, Tom Szymanski, and Chris Van Wyk for valuable comments of the paper. We are also indebted to Mely Chi, Shen Lin, Doug Schmidt, Jiri Soukup, and Manfred Wiesel for help and consultation during the development and implementation of this system, and to Dave Johnson for advice on simulated annealing.

References

[1] G. Persky, "PRO–An automatic string placement program for polycell layout," in *Proc. 13th Design Automation Workshop*, 1976.
[2] D. N. Deutsch, "A 'dogleg' channel router," in *Proc. 13th Design Automation Workshop*, 1976.
[3] U. Lauther, "A min-cut placement algorithm for general cell assemblies based on graph partitioning," in *Proc. 16th Design Automation Workshop*, June 1979.
[4] M. Burstein, "Partitioning of VLSI networks," in *Proc. 19th Design Automation Workshop*, 1982.
[5] D. C. Schmidt and L. E. Druffel, "An iterative algorithm for placement and assignment of integrated circuits," in *Proc. 12th Design Automation Workshop*, 1975.
[6] L. I. Corrigan, "A placement capability based on partitioning," in *Proc. 16th Design Automation Workshop*, 1979.
[7] M. A. Breuer, "Min-cut placement," *J. Design Automat. Fault Tolerant Comput.*, vol. 1, no. 4, pp. 343-362, Oct. 1977.
[8] C. Guinn and M. Breuer, "A force directed component placement procedure for printed circuit boards," *IEEE Trans. Circuits Syst.*, vol. CAS-26, June 1979.
[9] D. G. Schweikert, "A 2-dimensional placement algorithm for the layout of electrical circuits," in *Proc. 14th Design Automation Workshop*, 1976.
[10] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," in *Proc. 9th Design Automation Workshop*, 1972.
[11] L. Hyafil and R. L. Rivest, "Graph partitioning and constructing optimal decision trees are polynomial complete problems," Rep. 33, IRIA-Laboria, 1973.
[12] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291-308, 1970.
[13] B. W. Kernighan and S. Lin, "Method of minimizing the interconnection cost of linked objects," U.S. Patent 3 617 714, Nov. 2, 1971.
[14] M. Hanan, "Net wiring for large scale integrated circuits," IBM Tech. Rep. RC 1375 Feb. 1965.
[15] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," *Proc. 19th Design Automation Workshop*, 1982.
[16] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, May 13, 1983.

*



Alfred E. Dunlop (S'71-M'80) received the B.S.E.E. degree from the University of Delaware, Newark, in 1975 and the M.S. and Ph.D. degrees in electrical engineering from Carnegie-Mellon University, Pittsburgh, PA, in 1976 and 1979, respectively.

In 1977, he joined the Computer Aided Design Department at Bell Laboratories in Murray Hill, NJ. In 1981 he was on internship in a design laboratory. In 1982, he became Supervisor in The Computer Aided Design and Test Laboratory.

*

Brian W. Kernighan (S'62-M'72) received the Ph.D. degree in electrical engineering and compute science from Princeton University, Princeton, NJ.

He has been with the Computing Science Research Center at Bell Laboratories in Murray Hill, NJ, since 1969. His current research interests include document preparation, programming languages, and software tools. He is the co-author of several books, including *Software Tools, The C Programming Language*, and *The Unix Programming Environment*, and is on the editorial advisory boards of *Software Practice and Experience* and *The Bell System Technical Journal*.

Dr. Kernighan is a member of ACM.