A CLASS OF MIN-CUT PLACEMENT ALGORITHMS†

by

Melvin A. Breuer
Departments of Electrical Engineering & Computer Science
University of Southern California
Los Angeles, California 90007

Summary

In this paper we present a class of min-cut placement algorithms for solving some assignment problems related to the physical implementation of electrical circuits. We discuss the need for abandoning classical objective functions based upon distance, and introduce new objective functions based upon "signals-cut." The number of signals cut by a line c is a lower bound on the number of routing tracks which must cross c in routing the circuit. Three specific objective functions are introduced and the relationship between one of these and a classical distance measure based upon half-perimeter is presented.

Two min-cut placement algorithms are presented. They are referred to as Quadrature and Slice/Bisection. The concepts of a block and cut line are introduced. These two entities are the major constructs in developing any new min-cut placement algorithm.

Most of the concepts presented have been implemented, and some experimental results are given.

I. Introduction

This paper deals with a classical problem encountered in the physical implementation of circuit cards or chips, referred to as the placement problem. The problem is defined, semi formally, as follows. Given a set of elements $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ and a set of signals $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$. We associate with each element $e \in \mathcal{E}$ a set of signals $\mathcal{S}_e$, where $\mathcal{S}_e \subseteq \mathcal{S}$. Similarly with each signal $s \in \mathcal{S}$ we associate a set of elements $\mathcal{E}_s$, where $\mathcal{E}_s = \{e \mid s \in \mathcal{S}_e\}$. $\mathcal{E}_s$ is said to be a signal net. We are also given a set of slots or locations $\mathcal{L} = \{L_1, L_2, \ldots, L_p\}$, where $p \geq n$. The placement problem is to assign each $e_i \in \mathcal{E}$ to a unique location L_j such that some objective is optimized. Normally each element is considered to be a point, and if e_i is assigned to location L_j then its position is defined by the coordinate values (x_j, y_j). Usually a subset of the elements in $\mathcal{E}$ are fixed, i.e., pre-assigned to locations, and only the remaining elements can be assigned to the remaining unassigned locations. Non fixed elements are called moveable elements, and those slots not pre-assigned elements are called open slots.

Example placement problems deal with polycell LSI chip design [2,4,5], PC card design, and assigning functions to previously placed IC's.

For generality we will refer to the area (such as a chip, card or board) containing the slots or locations as the carrier.

After the elements are assigned to locations the resulting circuit configuration is routed. The automatic routing of 100% of the signals in $\mathcal{S}$ is one of the main objectives of the physical implementation portion of a design automation system.

Returning now to the placement problem we see that our actual goal is to assign each element to a location in such a manner as to maximize the routability of the resulting layout. However, this is a very intangible goal, since it is highly dependent on a number of factors, one being the basic algorithmic method employed by the router.

Let d_s be an estimate of the total distance of route required to interconnect net s, and set $N_d = \sum_{\forall s} d_s$. Then classical placement algorithms attempt to assign elements to locations such that N_d is minimized. By so doing it is felt that routability is increased.

Usually d_s is either the length of a minimal spanning tree (MST) for $\mathcal{E}_s$, one-half the perimeter of the minimal enclosing rectangle, denoted by $\frac{1}{2}P$, or the Steiner distance. The corresponding total distances using the MST, $\frac{1}{2}P$ or Steiner length are denoted by $N_d(MST)$, $N_d(P)$, and $N_d(S)$, respectively.

In the placement algorithm to be proposed in this paper we suggest a new objective function. This objective was motivated by two observations, namely (1) successful routing of a carrier is dependent on the density of interconnections on the carrier, and (2) some areas of a carrier are more dense than others.

Let c be a horizontal line crossing the surface of a carrier. For an arbitrary signal s, if one or more elements in $\mathcal{E}_s$ are above c, and one or more elements is $\mathcal{E}_s$ are below c, then when routing signal net $\mathcal{E}_s$ at least one connection must cross line c [9]. We say that line c is a cut line, and c is said to cut signal s. For a given plaement, the value of c, denoted by v(c), is the total number of signals cut by c. In general, v(c) is highly dependent on the location of c as well as the geometry of the carrier. In this paper we employ a family $\mathcal{C}_V$ and $\mathcal{C}_H$ of vertical and horizontal cut lines. Our objective function is to minimize some function f of the values of each cut line in $\mathcal{C}_V$ and $\mathcal{C}_H$. Placement procedures which minimize the value of such a function are called min-cut placement algorithms.

Druffel and Schmidt [1, 7] have briefly referred to a placement procedure that falls into our category of min-cut placement.

Also a brochure [6] describing a proprietory DA system makes reference to using this concept for placement.

II. Development of Objective Functions for Min-Cut Placement Algorithms

A. Classical Objective Function

The first objective function one might consider is the function

$$N_c(\sigma) = \sum v(c) \qquad (1)$$

†This work was supported in part by the National Science Foundation under Grant ENG74-18647.

where the sum is over all $c \in \mathcal{C}_V \cup \mathcal{C}_H$. To evaluate the properties of this function one must first define the location of the cuts in $\mathcal{C}_V$ and $\mathcal{C}_H$.

Consider a regular carrier geometry where elements are laid out in columns and rows, and where the distance between each column and row is one unit. We define a canonical set of cut lines as the collection of cut lines between each row and each column. Then the following result can be easily proven.

Theorem 1: Using a canonical set of cut lines, minimizing the objective function $N_c(\sigma) \triangleq \sum v(c)$, where the sum is taken over all cut lines, is equivalent to minimizing the objection function $N_d(P) = \sum d_s$, where the sum is taken over all signal nets and the measure taken for d_s is $\frac{1}{2}P$. ■ Hence, for an optimal placement $N_c(\sigma) = N_d(P)$. Since minimizing the function $N_c(\sigma)$ is equivalent to minimizing the function $N_d(P)$, using the function $N_c(\sigma)$ would not lead to any new results which could not be obtained by existing techniques.

B. Min-Max Objective Function

For some carrier technologies, such as polycell LSI carriers, a suitable objective function is

$$N_c(mM) = \min(\text{Max}\{v(c) \mid c \in C\}) \qquad (2)$$

where C is a set of cut lines. This objective function is most useful in trying to reduce the maximum track usage in a channel. Such an objective function is difficult to satisfy, and will not be dealt with in this paper.

C. Sequential Objective Function

We will next present an objective function, called a sequential objective function and denoted by $N_c(S_q)$, whose near minimal value is relatively easy to achieve and which also leads to good placements.

The form of this class of objective function is

$$N_c(S_q) = \min v(c_{i_r}) \mid \min v(c_{i_{r-1}}) \mid \cdots \mid \min v(c_{i_1}) \qquad (3)$$

where $c_1, c_2, \ldots, c_r$ is a given set of cut lines, $(i_1, i_2, \ldots, i_r)$ is a permutation ρ on $(1, 2, \ldots, r)$, "|" can be read as "subject to," and the expression for $N_c(S_q)$ is read from left to right. Hence $c_{i_1}, c_{i_2}, \ldots, c_{i_r}$ represents an ordered sequence of cut lines, c_{i_1} being the first cut line processed and c_{i_r} the last. By appropriate selection of ρ we can specialize the placement algorithm to fit a specific carrier geometry.

In the remainder of this paper we will discuss and illustrate heuristic procedures for minimizing $N_c(S_q)$, as well as present several useful orderings for the cut lines.

III. Basic Structure of Min-Cut Placement Algorithms

Consider the carrier shown in Fig. 1, where some arbitrary assignment of elements to slots has occurred. We refer to such an assignment of moveable elements to slot locations as a temporary assignment, denoted by T-assignment. A rectangular subarea A of the board is shown along with a cut line c. Line c divides A into areas A_1 and A_2. Now, within A_i, i=1, 2, we can re-assign some T-assigned elements located in A_1 to be in A_2, and an equal number of T-assigned elements in A_2 to be in A_1. Assume we do this in such a way so that the total number of signals on the carrier crossing c is minimized, subject of course to the constraint that elements not in A cannot be moved. This new assignment of T-assigned elements is again considered to be a T-assignment.

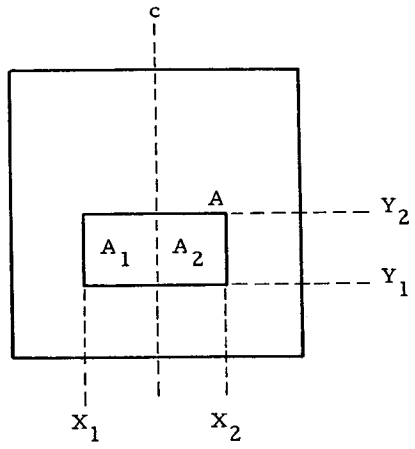

Figure 1. Carrier and a Block

The original area A, along with the slots and elements within A is called a block, and is denoted by B. Note that a cut line divides a block into two new blocks. The process of dividing a block into two new blocks is called block division. A moveable element is not considered placed until the block it is in has only one open slot. At this time it is said to be placed or assigned to that slot.

Let $\mathcal{B}$ be a set of disjoint blocks. Initially let the entire carrier be block B_1, and set $\mathcal{B} = \{B_1\}$. Then one form of a min-cut placement algorithm for mini-cut placement algorithm for minimizing $N_c(S_q)$ is as follows.

Algorithm 1: Cut oriented min-cut placement algorithm for $N_c(S_q)$.

1. Select a sequence (permutation ρ) for processing the cut lines.
2. Select next cut line c in sequence to process.
3. Assume c cuts across a subset of blocks $\mathcal{B}' = \{B_{i_1}, B_{i_2}, \ldots, B_{i_t}\}$. Re-assign T-assigned elements within these blocks such that v(c) is minimized.
4. In a natural way, from two new blocks from each of the blocks cut by c.
5. If there are no more cut lines to process, or if every block contains at most one moveable element, then exit, else return to step 2. ■

The sequence in which cut lines are to be processed can be either fixed or adaptive.

Example: Consider the carrier shown in Fig. 2a along with the five cut lines. Assume the lines are to be processed in the sequence c_1, c_4, c_5, c_2, c_3. Starting with the initial block $\mathcal{B}$, consisting of the entire carrier and all elements, we process B_1 with respect to c_1, hence minimizing $v(c_1)$. Block B_1 is now divided into two new blocks, denoted by B_1 and B_2 (this is a new block B_1). Note that once an element has been assigned to the left (right) of c_1 and B_1 is divided, the element can never be moved to the other side of c_1 no matter where subsequent cut lines occur. We now process B_1 and B_2 (simultaneously) with respect to c_4, producing the four blocks shown in Fig. 2d. Next we process

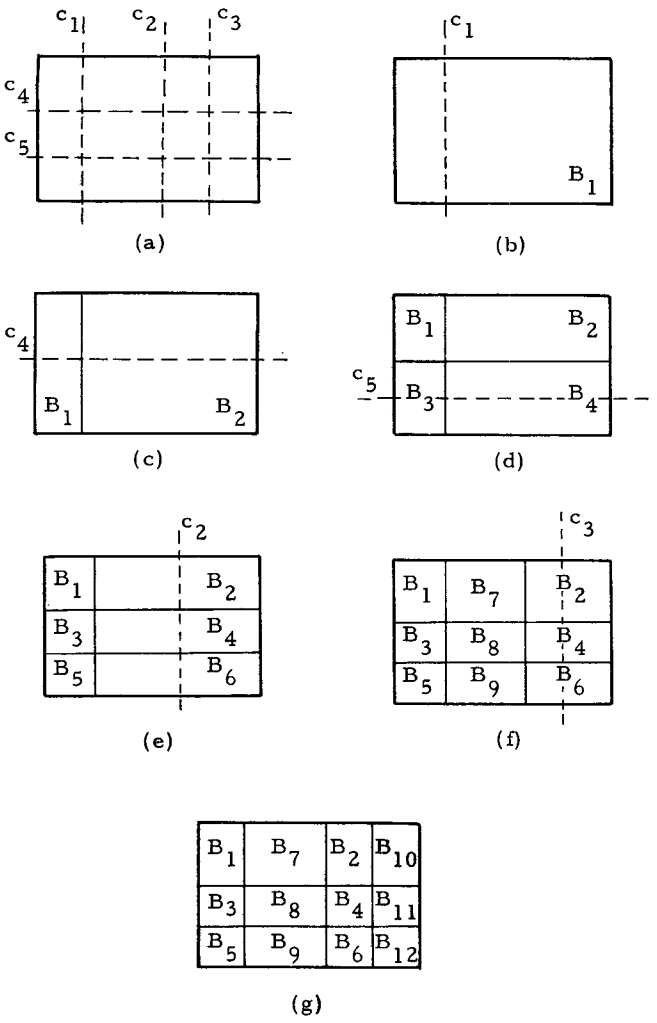

Figure 2. Min-Cut Algorithm 1 Processing B_1 with Respect to the sequence c_1, c_4, c_5, c_2, c_3

c_5 which only intersects B_3 and B_4. Note that even though the moveable elements in B_1 and B_2 are only T-assigned, the fact that we do not know optimal locations for these elements is immaterial since they are above c_4 hence can never be moved below c_5. Therefore the actual locations of these elements in B_1 and B_2 need not be known when calculating the value of c_5. Continuing the processing of the cuts we obtain the resulting 12 blocks as shown in Fig. 2g. ■

It is clear that this procedure realizes the objective function $N_c(S_q)$. In practice, it is not always desirable to execute Algorithm 1 exactly as specified. This occurs for the following two reasons. Consider Fig. 2c. In processing c_4 we must process B_1 and B_2 simultaneously. To reduce computation time it is advantageous to process B_1 and B_2 sequentially. That is, first process B_1 and then B_2. When processing B_1 we must ignore the moveable elements in B_2 since they may have been randomly assigned to slots. Once B_1 has been processed, that is, T-assigned elements re-assigned above and below c_4, we can now use this information in processing B_2. Once the elements in B_2 have been re-assigned to slots, we can now reprocess B_1 with respect to c_4 using this information. Hence we can iteratively process B_1 and B_2 until no new T-assignments occur. At this time the blocks can be divided. When dealing with large blocks this scheme will significantly decrease computation time. This occurs because the process of T-assigning elements is of complexity $\mathcal{O}(n!)$ for a block of n elements. Heuristic procedures for carrying out this process are approximately of complexity $\mathcal{O}(n^3)$. Hence, for example, rather than processing one block of n elements it is usually significantly faster to process, possibly several times, two blocks of size, say n/2.

In general, we have adapted the strategy of processing a set $\mathcal{B}'$ of blocks sequentially, and iteratively, rather than simultaneously. The disadvantage of this type of processing is that we may achieve a local minimal rather than a global minimal.

Another modification to the procedure just presented deals with the problem of where to cut a block. One useful criteria is to cut a block so that half of the open slots are on either side of the cut line. This is called bisecting. Again consider blocks B_1 and B_2 (Fig. 2c). Assume B_1 has many preassigned elements above c_4 while B_2 does not. Then bisection of B_1 and B_2 cannot be done by the same cut line. If we select different cut lines for B_1 and B_2, then we are no longer dealing with the simple sequential objective function given by eq. (3).

We see that it is possible to dynamically select cut lines as one proceeds, based upon the location of fixed and open slots in the various blocks encountered. Next we present a revised version of our initial algorithm which allows for this dynamic handling of cut lines. For this algorithm we can no longer write our objective function in a simple concise form.

Algorithm 2: Block oriented min-cut placement algorithm.

1. Select next block B (or set of blocks $\mathcal{B}'$) for division.
2. Select cut lines for these blocks.
3. Re-assign moveable elements in B or $\mathcal{B}'$* such that the number of signals cut is minimized.
4. Subdivide blocks, forming a new set of blocks $\mathcal{B}$.
5. If all blocks now contain a single moveable element then exit, else return to step 1. ■

Block Selection

Blocks can be processed in any order. Two natural orderings are referred to as depth first and breadth first.

These orderings can be realized by storing new blocks on a stack. Breadth first is realized with a first-in first-out stack. In breadth first we normally process one or more blocks at a time, while in depth first we process only one block at a time.

Selection of Cut Lines

Normally the location of each allowable cut line can be specified in advance, e.g., between rows and/or columns of slots. The actual problem of selecting a cut line is thus in determining in what sequence to process these lines. The selection criteria is usually a function of the carrier geometry and predicted routing density. For fixed ordering we have found that two types of cut lines are quite effective; they are referred to as a slice cut and a bisection cut. A slice cut c of a block B is a cut line which isolates a fixed number (K) of slots of B to one side of c and the remaining slots to

*The blocks in $\mathcal{B}'$ are sequentially processed, and if desired, iteratively processed.

the other side of c. A bisection cut c of a block is a cut line which tends to evenly divide the unassigned elements in B to either side of c. By "tends to" we mean that there exists no other cut line c′ which more closely divides the unassigned elements.

IV. Two Fixed Sequential Min-Cut Placement Algorithms

In this section we will describe two specific min-cut placement procedures corresponding to Algorithm 2. They differ in the order in which blocks are processed and the type of cut lines employed. We refer to these algorithms as Quadrature placement and Slice/Bisection placement.

A. Quadrature Placement Procedure

In this procedure the original block (carrier) B_1 is first bisected by a vertical cut line producing two new blocks B_1 and B_2. These two blocks are then cut by horizontal bisecting cut lines thus forming up to four blocks. These blocks are then cut by vertical bisecting cut line. This process is repeated, alternating between vertical and horizontal cut lines, until each element is placed. Note that this procedure processes blocks breadth first. The quadrature placement algorithm is designed for carriers having a high density of routing in their center. By first processing cut lines in the center of the carrier we attempt to push interconnections away from this region, and hence produce a placement which can be routed with a more uniform density.

B. Slice/Bisection Placement Procedure

In this procedure (see Fig. 3), we first divide the initial set of n elements into a set of K and n-K elements, where K> 0. Again v(c) is minimized. These K elements represent the bottom row or slice of components on a carrier. This procedure is repeated on the remaining (n-K) elements, again dividing them into a set of K elements, and a set of (n-2K) elements. This process is repeated until all elements have been assigned to a row. The elements are then assigned to columns via vertical bisecting.

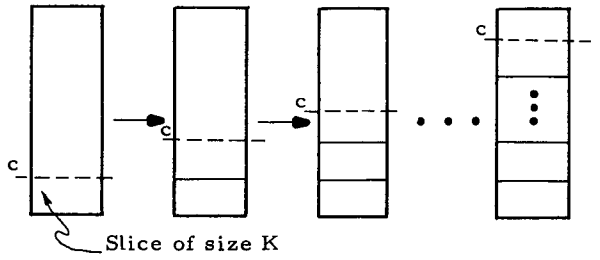

Figure 3. Slice/Bisection Placement Scheme — Growing Horizontal Slices

This technique is best suited to carriers where there is a high interconnect density at the terminals.

V. Formalization and Implementation Aspects of Min-Cut Placement Procedures

A. A Block

In this section we present a more detailed version of the placement procedures described in Sec. IV. Recall that the basic structure behind a min-cut placement procedure is that of a block. Intuitively a block is a section of a carrier containing some fixed (assigned) elements and some moveable (unassigned) elements. A block can be divided (partitioned) by a cut line into two blocks. Formally, a block B_i consists of a 7-tuple $\langle Z_i, FE_i, FLOC_i, f_i, ME_i, ALOC_i, F_i \rangle$ defined as follows:

1. $Z_i = (X_1^i, X_2^i, Y_1^i, Y_2^i)$ – the coordinates defining the physical boarders associated with B_i (see Fig. 1).
2. FE_i – a finite set of fixed elements located within Z_i.
3. $FLOC_i$ – a set of locations each of which is within Z_i, where $|FE_i| = |FLOC_i|$.
4. f_i – a function assigning each element in FE_i to a unique element in $FLOC_i$.
5. ME_i – a set of moveable (unassigned) elements in B_i.
6. $ALOC_i$ – a set of available locations in Z_i for placing the elements in ME_i. We assume that $|ME_i| = |ALOC_i|$. If there are actually more locations than elements, then dummy elements are defined. (Contrary to the previous discussion, we do not actually employ a mapping of elements in ME_i to elements in $ALOC_i$, i.e. we just think of the moveable elements and available locations as two sets.)
7. F_i – a flag which indicates whether or not a block has been processed. This flag is used when a set of blocks must be iteratively processed.

A vertical cut line c of block B is said to define two pseudo blocks B′ and B″ where B′ and B″ are defined as follows. In this definition c has x-coordinate X_0, where $X_1 \leq X_0 \leq X_2$.

1. $Z' = (X_1, X_0, Y_1, Y_2)$ and $Z'' = (X_0, X_2, Y_1, Y_2)$.
2. FE′ (FE″) is the subset of elements in FE to the left (right) of c.
3. FLOC′ (FLOC″) is the subset of slots in FLOC to the left (right) of c.
4. f′ (f″) is the restriction of f to the domain FE′ (FE″).
5. ME′ (ME″) is the subset of elements in ME to the left (right) of c.
6. ALOC′ (ALOC″) is the subset of locations in ALOC to the left (right) of c.
7. FLAG′ (FLAG″) – flags associated with B′ and B″ initially set to 0.

B′ and B″ are said to be pseudo blocks since their moveable elements can be re-assigned from B′ to B″ and vice versa. That is, they can be re-assigned from one side of c to the other.

A vertical assignment of a block B consists of

a. determining a vertical cut line c for B;
b. constructing pseudo blocks B′ and B″; and
c. assigning the elements of ME to ME′ and ME″

such that some objective is minimized, such as the number of signals cut by c.

A vertical assignment of a set of blocks $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ consists of a vertical assignment of each $B_i \in \mathcal{B}$.

A vertical bisection assignment is a vertical assignment generated by a vertical bisection cut line.

B. Partitioning

The key problem in carrying out a vertical assignment is that of assigning the elements in ME to the sets ME′ and ME″ such that we minimize the number of signals cut by c. This problem is a generalization of the following partitioning problem. Given a graph G having n nodes, partition the set of nodes of G into two

disjoint sets N_1 and N_2 of nodes having n_1 and n_2 elements respectively, where $n_1+n_2=n$, and such that the number of edges between N_1 and N_2 is minimal. Kernighan and Lin [3] have described a heuristic procedure which appears to produce very fine results for n large. Their procedure starts with an initial (arbitrary) partition of N into sets N_1 and N_2, and computes sets $A \subseteq N_1$ and $B \subseteq N_2$, $|A| = |B|$ such that interchanging A and B reduces the number of edges between the resulting set of nodes to a minimum. The resulting partition of N is thus $N_1' = N_1 - A + B$ and $N_2' = N_2 - B + A$, where "−" and "+" refer to set operations. The procedure can then be repeated starting with N_1' and N_2' as the initial partition. This iterative procedure is halted according to some user specified termination rule.

To apply the Kernighan-Lin procedure to our problem we must extend it two ways. First instead of cutting an edge between two nodes we must deal with cutting signals between sets of nodes. This extension is quite trivial to implement and has been previously discussed by Schweikert and Kernighan [8]. The second extension required is that we must restrict some elements in N_1 and N_2 to be unavailable for interchange. These elements in N_1 and N_2 correspond to our fixed elements. The extension of the Kernighan Lin procedure which includes these two generalizations is referred to as the Generalized Kernighan-Lin procedure, denoted by GK-L. We define a GK-L subroutine as follows:

Subroutine GK-L ($F', F'', M', M'', M^*, M^{**}$).
Input parameters: Fixed set of elements F' and F'' and moveable set of elements M' and M''.
Output parameters: Two moveable sets of elements M^* and M^{**}, where $M^*, M^{**} \subseteq M' \cup M''$, $M^* \cup M^{**} = M' \cup M''$, $M^* \cap M^{**} = \varphi$, and $|M'| = |M^*|$ and $|M''| = |M^{**}|$.
Function: M^* and M^{**} are computed using the generalized Kernighan-Lin procedure such that the number of signals between the set of elements $\{F', M^*\}$ and $\{F'', M^{**}\}$ is minimal, where the initial partition is $\{F', M'\}$ and $\{F'', M''\}$.

We can now employ the GK-L subroutine to compute a vertical bisection assignment for a set of blocks $\mathcal{B}' = \{B_1, B_2, \ldots, B_q\} \subseteq \mathcal{B}$. Next we describe a heuristic procedure for creating this assignment where the objective is to minimize the total number of signals cut. This procedure has two modes of operation, namely iterative or noniterative. In the iterative mode the blocks in $\mathcal{B}'$ are repeatedly processed until no new reductions in the cut values occurs.

Algorithm 3: Vertical-Bisection Assignment

Step 1: For $i = 1, 2, \ldots, q$ set $F_i = 0$, and construct a vertical bisect line c_i for B_i. If c_i does not exist, so note. The x-coordinate of c_i is X_0^i. Set FIRST = 1. Read in MODE (iterative or non-iterative).

Step 2: For $i = 1, 2, \ldots, q$ do the following: (V bisection assignment of B_i)

Step 2.1: If c_i exists go to step 2.2, else set $F_i = 2$ and return to step 2.1 for next i.

Step 2.2: (Construction of F' and F'')
a) For each fixed element e in any block of $\mathcal{B}$ to the left (right) of X_0^i, put e into F' (F'').
b) For each block B_j such that $X_2^j \leqq X_0^i$ ($X_2^j \geqq X_0^i$), put all moveable elements into F' (F'').
c) For every block B_j such that $F_j \neq 0$ and X_0^i cuts (intersects) B_j, if $X_0^i \leqq X_0^j$ set all elements in ME_j' into F', otherwise set all elements in ME_j'' into F''.

Step 2.3: If FIRST=1 go to step 2.4 else go to step 2.5.

Step 2.4: (Construction of initial M' and M'') Partition ME_i into disjoint sets having $|ALOC_i'|$ and $|ALOC_i''|$ elements each, and assign these elements to M' and M'' respectively using an initial partition algorithm.* Go to step 2.6.

Step 2.5: (Construction of M' and M'') Set $M' = ME'$ and $M'' = ME''$. (Here we use the old value of ME' and ME'' for the initial value of M' and M''.)

Step 2.6: Call subroutine GK-L ($F', F'', M', M'', M^*, M^{**}$).

Step 2.7: Set $ME_i' = M^*$ and $ME_i'' = M^{**}$ (i.e. and "optimal" assignment of ME_i has been made such that $v(c_i)$ is minimized). Set $F_i = 1$ if $M^* = M'$, else $F_i = 2$. ($F_i = 2$ implies a new assignment of B_i has been computed, i.e. $M^* \neq M'$, while $F_i = 1$ implies no new assignment has occurred.

Step 3: MODE = iterative?

Yes: Set FIRST = 0. If $F_i = 2$ for some i, set $F_i = 0$ for all i and go to step 2, otherwise EXIT.

No: EXIT.

This procedure is finite since whenever a block is processed, either v(c) is reduced or else $M' = M^*$. Since v(c) cannot be indefinitely reduced, eventually $M_i' = M_i^*$ for all i and we exit the procedure.

Once $\mathcal{B}'$ has been processed by Algorithm 3, each pseudo block is made into a block. For example, the pseudo block B_i' defined by the 7-tuple $\langle (X_1^i, X_0^i, Y_1^i, Y_2^i), FE_i', FLOC_i', f_i', ME_i', ALOC_i', F_i \rangle$ now defines a new block B_j, defined by the parameters $(X_1^j, X_2^j, Y_1^j, Y_2^j) = (X_1^i, X_0^i, Y_1^i, Y_2^i)$, $FE_j = FE_i'$, $FLOC_j = FLOC_i'$, etc. We refer to the process of defining pseudo blocks as real blocks as block division. Hence, by carrying out block assignment followed by block division a set $\mathcal{B}' = \{B_1, B_2, \ldots, B_q\}$ is transformed into a new set of blocks $d(\mathcal{B}') = \{B_1', B_2', \ldots, B_m'\}$, where $m \leqq 2q$. Note that since some blocks do not have a bisect line, m can be less than 2q.

In a completely analogous manner to the preceding discussion, we can define the corresponding block assignment and division procedures for horizontal cut lines.

C. Placement Algorithms

The procedures for vertical and horizontal bisection or slice assignment followed by block division make up the main routines in our Quadrature and Slice/Bisection placement algorithms.

As an example, consider the set of blocks $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ shown in Fig. 4a. If we vertically slice assign B_n and then divide the resulting block we obtain the

*This is a constructive partitioning procedure which procudes much better final results than when a random assignment is used.

set of blocks $\mathcal{B} = \{B_1, B_2, \ldots, B_{n+1}\}$ shown in Fig. 4b. We call this process a vertical slice division of $\mathcal{B}$. Note that the vertical slice assignment of B_n can be obtained by a simplified form of Algorithm 3, namely by setting $\mathcal{B}' = B_n$, employing a slice cut rather than a bisectional cut, and by setting the MODE to non-iterative.

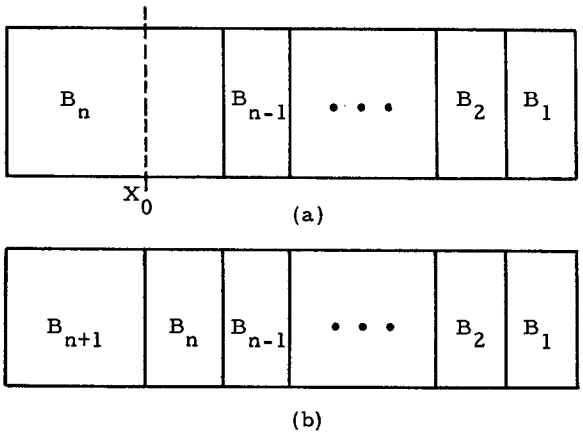

Figure 4. Development of a Slice
(a) Before Slice
(b) After Slice

VI. Experimental Results

Some of the concepts described in this paper have been implemented, and in this section we will briefly describe a few of the results obtained.

The programs are written in PL/I and run on an IBM 370/165.

All the problems to be discussed in this section deal with the placement and routing of PC cards. The carriers are 5" x 5", consist of 5 columns and 10 rows of 14 or 16 pin IC DIPS, and have two signal layers for interconnection. We thus have a density of 2 IC's/in^2. Routing is carried out on 25 mil centers, hence a very high routing density exists.

The physical design of the card is usually carried out in the following order. All steps not modified by the work "manual" are done automatically.

The process begins with an initial assignment of logic functions (elements) to IC's. This step can be done either manually by the logic designer or automatically by a constructive assignment algorithm. After this step we iteratively use the min-cut placement procedures to first place IC packages and then re-assign elements to packages. The same program can carry out both functions. At the conclusion of this step a constructive placement procedure is employed to assign each element assigned to an IC to the optimal portion of an IC. This step is then followed by a constructive procedure which assigns signal nets to IC pins. Finally the card is automatically routed via a Lee type routing algorithm.

In Table 1 we summarize the results for three "difficult" cards using our slice/bisection algorithm. For card no. 1 we see that the given manual placement has a half-perimeter ($N_d(P)$) of 838 (inches), and when routed produced 37 failures (signal nets not successfully routed). This same card was then processed using automatic placement. The initial random placement used by the system has a half-perimeter of 888. Three different runs are documented in Table 1. The first employs the min-cut placement algorithm once on the IC's. Run 2 employs this algorithm twice on IC's and once on the elements. Run 3 employs this procedure three times on the IC's and twice on the elements.

	Manual Placement Automatic Routing		Automatic Placement and Routing	
Card	$\frac{1}{2}$P	# of Failures	$\frac{1}{2}$P	# of Failures
1	838	37	Run #1 658	19
			Run #2 646	15
			Run #3 634	11
2 289 nets	-	28	554	5
3 233 nets	655	35	616	6

Table 1: Placement/Routing Experimental Results

Finally, cards 2 and 3 show similar improvement in routing due to this placement program.

To process a board through initial placement of elements to packages, two passes through IC placement and one pass through element re-assignment, gate to IC portion assignment, and IC pin assignment, requires, on the average, 50 CPU seconds at a prime time cost of $25 per minute.

We have found from experience that if $\frac{1}{2}$P is less than 620, we obtain almost no failures. Hence we usually iterate the element - IC placement procedure until either we reach a $\frac{1}{2}$P value of less than 620, or else if we see that no significant improvement in the placement is possible. We then go into our routing phase.

We have also carried out numerous experiments dealing with the order in which cut lines should be processed, and have found that Quadrature is usually the best technique for our board geometry, where we first carry out a vertical cut (perpendicular to the I/O connector).

References

1. L. E. Druffel, D. C. Schmidt, and R. A. Wagner, "A simple, efficient design automation processor," Proc. 11th Design Automation Workshop, June 1974, pp. 127-136.
2. B. W. Kernighan, D. G. Schweikert, and G. Persky, "An optimal channel routing algorithm for polycell layouts of integrated circuits," Proc. D. A. Workshop, 1973, pp. 50-59.
3. B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell Sys. Tech. J., vol. 49, February 1970, pp. 291-308.

4. R. L. Mattison, "Design automation of MOS artwork," Computer, vol. 7, January 1974, pp. 21-27.

5. G. Persky, D. N. Deutsch, and D. G. Schweikert, "LTX-a system for the directed automatic design of LSI circuits," Proc. Design Automation Conference, June 1976, pp. 399-407.

6. "PRANCE," (brochure), Automated Systems, Inc., 999 Sepulveda Blvd., El Segundo, Calif. 90245.

7. D. C. Schmidt and L. E. Druffel, "An iterative algorithm for placement and assignment of integrated circuits," Proc. 12th Design Automation Conference, June 1975, pp. 361-368.

8. D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," Proc. Design Automation Workshop, June 1972, pp. 56-62.

9. I. E. Sutherland and D. Oestreicher, "How big should a printed circuit board be?" IEEE Trans. on Computers, vol. C-22, May 1973, pp. 536-541.