Switchbox Routing by Stepwise Reshaping

SABIH H. GEREZ AND OTTO E. HERRMANN, MEMBER, IEEE

Abstract—**An algorithm for switchbox routing, called PACKER, is presented. In an initial phase, the connectivity of each net is established in a fast way without taking the other nets into account. In general this will give rise to conflicts (short circuits). In the second stage the conflicts are removed iteratively using *connectivity preserving local transformations* (CPLT's). They "reshape" a net by displacing one of its segments without disconnecting it from the net. The transformations are applied in a systematic way using a scan-line technique. During this process, a subset of the segments at the position of the scan line are densely packed in the (two) layers available for routing. The remaining segments are pushed to the next scan-line position. Scanning in the four available directions (from left to right, right to left, top to bottom, and bottom to top) is performed until all conflicts have disappeared or no solution is found within a maximum number of iterations. It turns out that the new approach to routing, as implemented in PACKER, also has practical merits: most of the well-known "benchmark" examples are solved.**

I. Introduction

THIS article explains the PACKER algorithm for switchbox routing. Switchbox routing is a well-known problem in the field of automatic layout for integrated circuits (see, e.g., [1]). The problem is to interconnect a number of nets within a rectangular area of fixed size, called a switchbox. The terminals are allowed to enter the switchbox from any of the four sides. In this article a switchbox routing problem is considered solved when, for each net, all its terminals have been interconnected inside the given area without having short circuits with other nets. Obtaining solutions of good *quality*, i.e., with total wire length and number of vias close to their minimal values, is only a secondary goal.

The switchbox routing problem is a special case of the *local* routing problem (also called *detailed* or *final* routing). A straightforward approach to reduce the complexity of local routing is to route one net at a time. In this context, *Lee-routing*, or *maze routing* [2] is a powerful tool to route a single net. However, its degree of success depends on the order in which the nets are processed. Sometimes the route chosen for a net can block the routing of a next one. To overcome this type of problem, *rip-up and reroute*-techniques are used [3].

In the recent past this situation was considered unsatisfactory by some researchers and their efforts resulted in new algorithms in which all nets are routed simultaneously (see, e.g., [4]–[6]). In these algorithms the routing steps are guided by the *interaction* of nets.

PACKER deals with the interaction of nets in a very natural way. The algorithm operates on a configuration in which all nets are always connected. Information on which nets occupy the same locations is, therefore, directly available. This information is used to reshape one of the two nets taking part in a *conflict* (short circuit), in order to remove the conflict.

A quick routing algorithm is used to obtain an initial configuration; it routes all the nets independently of each other. The idea behind reshaping is to modify as least as possible of the actual routing. A so-called *connectivity preserving local transformation* (CPLT) moves a single segment of the routing one grid position aside. What remains to be done, is to apply the CPLT's in a systematic way. PACKER uses a scan-line technique: all segments under the scan line are considered at the same time and a conflict-free subset of them is selected to remain; the rest have to move to the next scan-line position. Scanning in different directions (from left to right, right to left, top to bottom, and bottom to top) continues until all conflicts have been resolved or some time-out condition is met.

PACKER uses a routing model with the following properties:

1) routing is performed on an orthogonal grid;
2) two layers of wiring are used;
3) horizontal and vertical connections are allowed in both layers;
4) the terminals all have a fixed position and the layer of a terminal is fixed, i.e., a terminal has to be extended in this layer for at least one grid unit into the switchbox. However, all terminals on the same side do not need to enter the switchbox in the same layer.

This article is organized as follows: in the next section the approach of PACKER is compared to other approaches reported in literature. Then the algorithm is introduced in a bottom-up fashion. After explaining the top level, attention is paid to those aspects, the explanation of which was postponed for the sake of clarity. Finally, the results achieved are discussed. It turns out that PACKER succeeds in solving many well-known benchmark examples for switchbox routing.

II. Relation to Other Work

Most of the successful routers reported recently [5], [7]–[11] generate intermediate configurations which are free of conflicts at the expense of incompletely connected nets.

Manuscript received February 8, 1989. The work presented here was part of the project "Innovative CAD Tools for IC Design," supported by the Foundation FOM(TEL 330408). This paper was recommended by Associate Editor A. Dunlop.

The authors are with the Faculty of Electrical Engineering, University of Twente, 7500 AE Enschede, The Netherlands.

IEEE Log Number 8930757.

In PACKER, as well as in some algorithms that were developed independently at around the same time [12] (and also [13], but less strictly), the nets are always connected at the expense of conflicts.

The idea of allowing conflicts in intermediate configurations is not new. Rubin, in an algorithm for printed circuit boards (PCB's) [14], uses the information on conflicts to update cost factors in a Lee-type router (see, e.g., [3]). The same idea also can be found in more recent algorithms by Rosenberg [15] and Linsker [16]. Lin, Hsu, and Tsai [12] add a probabilistic element to a similar approach and succeed in applying the method to switchbox routing.

The use of what has here been called "CPLT's" is mentioned in [17], in an application for global routing, calling them "parallel displacements of the straight sections." No details are given, however, on *how* to get rid of overcongested channels (the counterpart of a conflict in global routing) in a systematic way. The "weak modifications" of [10] and the "wire pushing operations" of [13] have some similarities with CPLT's. In both cases, however, they are not the only methods for the modification of the wiring: Lee-routing, rip up, and reroute are also used. PACKER does not rely on anything else but the CPLT's.

A work which is quite close in philosophy to the ideas behind PACKER is the one reported by Fisher [18]: it also starts with with an independent routing of the nets and then tries to remove conflicts iteratively by means of transformations. However, the strategy for conflict resolution has strongly been influenced by the peculiarities of PCB routing.

III. The Main Data Structures of Packer

3.1. The Representation of a Route

The way a net has been routed is represented by two types of data structures: **wire-nodes** and **wire-segments**. The nodes represent geometrical points which are interconnected by segments. Pairs of pointers (from a node to a segment and vice versa) are used to realize the interconnections, as is shown in Fig. 1.

Whenever the segments connected to a node are in different layers, the node implicitly represents a contact cut. Otherwise, it is simply part of a wire. Although there are two layers, the structure of nodes and segments is two-dimensional. The layer is an attribute of a segment. The nodes on the boundary of the switchbox have a special attribute marking them as *terminal nodes*; segments attached to these nodes are called *terminal segments*.

3.2. Detection of Conflicts

Each segment "occupies" the grid points associated with the nodes connected to it and the grid points in between. When a grid point is occupied by segments belonging to different nets and the segments are in the same layer, there is a conflict. For the easy detection of conflicts each segment in the configuration is stored in two

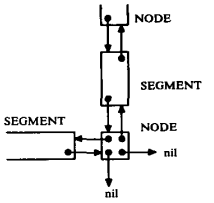


Fig. 1. A fragment of the data structure representing a route.

data structures, both based on McCreight's *priority search trees* [19].

1) A data structure for the detection of conflicts between perpendicular segments using the *on-line segment intersection algorithm* is described in [20]. The data structure used is a binary tree having a priority search tree in each node. Horizontal segments are stored in one tree and vertical ones in another tree.

2) A data structure for the detection of conflicts between overlapping parallel segments (*on-line intersection in a dynamic set of linear intervals* [19]). This structure has a priority search tree for each (horizontal and vertical) grid line.

The data structures are used to maintain up-to-date two fields for each segment in the switchbox:

1) a list of perpendicular segments with which the segment shares a grid point;
2) a list of parallel segments with which the segment shares a grid point.

IV. Initial Routing

The goal of initial routing is to connect all the nets quickly and independently of each other. In PACKER two different methods have been implemented. For both, the starting position is a situation where terminal segments of length one have been laid out (see Fig. 2(a)); the inner end points (wire nodes) of these segments have to be interconnected. This is done using one of the following methods.

1) Quasi-Minimal Steiner Tree: In this method, first a minimal *spanning* tree is computed using the algorithm of Prim [21]. Then the tree is used to actually interconnect the nodes: the branches of the tree determine the pairs of nodes to be interconnected. There is only one way of interconnecting when the two nodes are positioned on the same (horizontal or vertical) grid line. Otherwise, one of the two possible L-shaped interconnections are made. The connection which reuses most of the existing wiring for this net is chosen (cf. the algorithms described in [22], where the choice does not depend on the rest of the wiring). Fig. 2(b) shows the result for a simple example. What is obtained is an *approximation* of the *minimal rectilinear Steiner tree*, which is good enough for the purposes of PACKER, as the nets will be reshaped anyway. Considering that the computation time is orders of magnitude longer in the iterative part of PACKER, this approximation algorithm with quadratic time complexity has

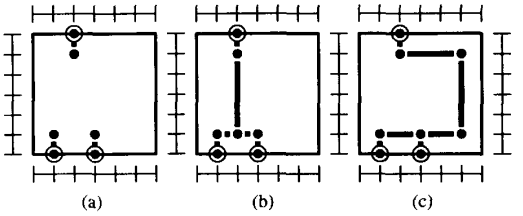


Fig. 2. The generation of the initial routing. (a) Starting point. (b) Quasi-minimal Steiner tree routing. (c) Counterclockwise routing.

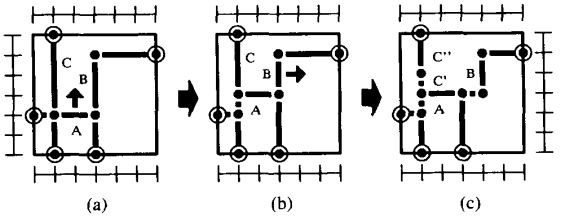


Fig. 3. Some simple moves: starting from (a) moving segment A gives (b) and moving B together with the splitting of C gives (c).

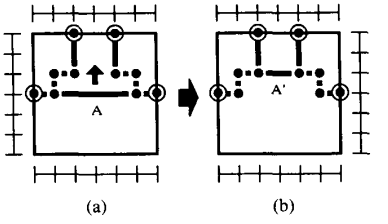


Fig. 4. Merging: (a) segment A becomes the shorter segment (b) A'.

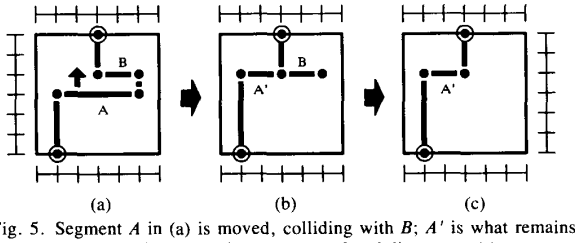


Fig. 5. Segment A in (a) is moved, colliding with B; A' is what remains after merging (b); B has a loose end and disappears (c).

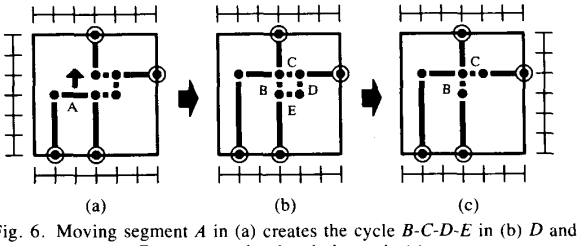


Fig. 6. Moving segment A in (a) creates the cycle B-C-D-E in (b) D and E are removed to break the cycle (c).

been chosen above a linear time algorithm; that solves the problem exactly [23], because of its simplicity.

2) Counterclockwise Routing Along the Boundary: The wire segments are always laid out along the outermost track available in the switchbox, going counterclockwise, starting from the bottom left and connecting all terminals encountered until the last one is met. Fig. 2(c) illustrates this method.

PACKER succeeds in finding solutions with both ways of initial routing as a starting point, although the second method, in general, asks for a higher computational effort. More details will be given in Section XII.

V. Connectivity Preserving Local Transformations (CPLT's)

CPLT's are the elementary moves in the reshaping of nets. Three different types of CPLT's exists:

1) moving a line segment a single grid position aside;
2) splitting a line segment longer than one grid unit into two line segments;
3) moving a segment to the other layer (this does not modify the pointers in the data structure, but only the layer attribute of a segment).

Simple examples of the first type, together with an example of splitting are shown in Fig. 3. When a segment is moved, the principle used is that as least as possible of the routing of the net should be affected. In simple cases only the perpendicular segments at the end points of the segment to be moved are made shorter or longer, they are split or newly created to maintain connectivity.

The situation becomes more complicated when the destination area already contains segments of the same net. In such a case, illustrated in Fig. 4, the segment to be moved has to be *merged* with those already present. Sometimes merging gives rise to a *loose end*, a wire segment with an end point which is not connected to any other segment, without being a terminal. Such a segment has to be removed, as illustrated in Fig. 5. In some cases moving a segment to the next grid position might even create a cycle in the wiring. In a cycle there is at least one segment that can be removed without losing the connectivity of the net. PACKER removes the longest sequence of segments in the cycle that does not disconnect the net (see Fig. 6) (if the longest sequence is not unique, an arbitrary choice is made). Note that in all moves requiring merging, the original configuration cannot be recovered in a single step: it might be said that moves of this type are "irreversible."

VI. Scanning

This section deals with the most essential part of PACKER, the packing of line segments in two layers while moving with a scan line across the switchbox. During scanning the segments present at the scan-line position are selected either to remain or to move. Those which have to remain are assigned a layer; those which have to move are "pushed" to the next scan-line position by means of CPLT's. During one scan the procedure that performs the selection is called first for the grid line next to one of the boundaries and ultimately for the second grid line next to the opposite boundary (see Fig.7). The con-

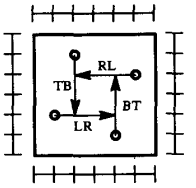


Fig. 7. The ranges of the four different types of scan: top-bottom (TB), right-left (RL), bottom-top (BT), and left-right (LR).

flicts accumulate at the first position next to this opposite boundary. A detailed description of the procedure executed for each scan-line position is given below.

1) Find PA, the set of segments parallel to the scan line at the scan-line position (see Fig. 8(b)).

2) Remove from PA the terminal segments; if there are two of them and they are in the same layer, they might have a conflict. A conflict between terminals is given special treatment. It is solved either by moving the segments connected to the terminal end points, if there are no *constraint relations* preventing the moves, or by splitting the terminal segments, when the constraints prohibit moving (see Fig. 9). Constraint relations will be explained in Section VIII.

3) Find PE, the set of perpendicular segments crossing the scan line (see Fig. 8(c)).

4) Although it need not be true in general, the following two assumptions are made: "the area behind the scan line is free of conflicts" and "the segments in PE have a correct position and will not be moved." The segments in PA will have to adapt to them. The segments in PE are partitioned into three groups: those of *fixed layer*, of *floating layer*, and the *postponed* segments. Fixed layer segments are called such because the conditions behind the scan line leave no choice for the layer of the segment. Floating layer segments can be assigned any layer without introducing conflicts behind the scan line, sometimes at the expense of a contact cut just behind the scan line. A postponed segment is connected to a segment s in PA and runs from the scan-line position to the part of the switchbox which still has to be visited by the scan line. When s is moved, the postponed segment will become shorter and will be out of the scope of the scan line. For this reason it is ignored until s is selected remain. Examples of the three different types of perpendicular segments are given in Fig. 10.

A fixed layer segment that crosses a terminal segment causes a conflict that cannot be solved in the current scan, because the terminal cannot move by definition and moving the perpendicular segment would violate the assumption made above. This segment is ignored and the conflict will be dealt with later, when scanning in a direction perpendicular to the current one.

5) Inspect the segments in PA, using the information on the partitioning of PE. If a segment is crossed by two fixed layer segments in different layers, the segment cannot remain at the scan-line position: it is removed from PA and added to M, the set of segments to be moved. If

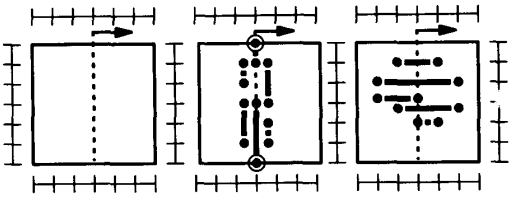


Fig. 8. (a) The scan line. (b) The parallel segments in the set PA (although they all lie on the same grid line, they have been drawn next to each other for the sake of clarity). (c) The perpendicular segments in the set PE.

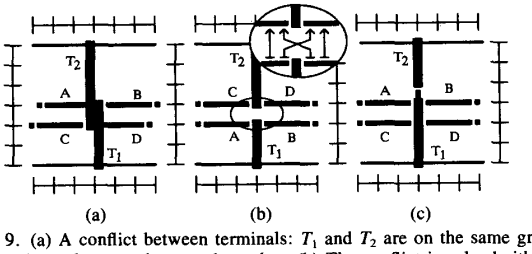


Fig. 9. (a) A conflict between terminals: T_1 and T_2 are on the same grid line, have the same layer and overlap. (b) The conflict is solved either by moving the perpendicular segments A, B, C, D, (c) or by splitting the terminals.

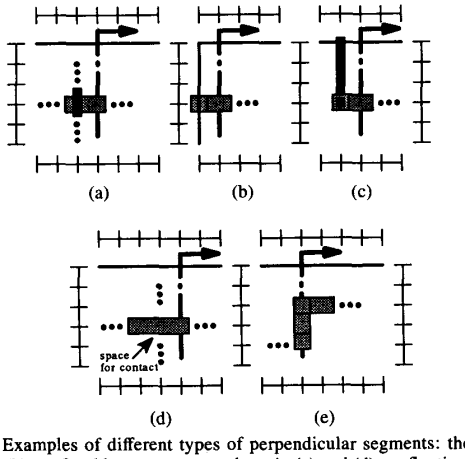


Fig. 10. Examples of different types of perpendicular segments: those in (a) and (b) are fixed layer segments; those in (c) and (d) are floating layer segments; a postponed segment is shown in (e).

a segment is crossed by fixed layer segments of the same layer, the segment, if selected to remain at the scan-line position, will have to be in the opposite layer; the layer change is effectuated immediately (see Fig. 11 for examples). So, after this step PA does not contain a segment anymore which is incompatible with the segments in PE.

6) Layer Propagation: Suppose that a segment s from PA has been selected to remain. This affects both the segments in PE as well as those in PA as far as they share a grid point with s. The partitioning in PE can change: floating layer segments become fixed layer segments and postponed segments attached to s become fixed or floating layer segments. Segments in PA can become incompati-

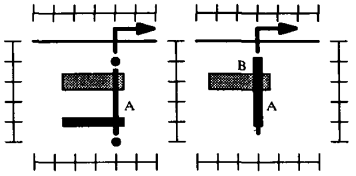


Fig. 11. Two examples of wires in PA. (a) Segment A has to move. (b) A will have to be in B's opposite layer.

ble with s and will have to move to M or they might have to change layer to remain compatible.

7) Call the layer propagation routine for the terminal segments, if present.

8) Assign a weight to all the segments in PA: the higher the weight, the more desirable it is to keep the segment in the current position. The different effects contributing to the total weight are discussed in detail in Section IX.

9) The problem which remains is to select from PA a maximum weight subset of segments that are compatible with each other, with the segments of PE, and also with the terminal segments. This problem is a weighted and constrained variant of the graph theoretical *maximal independent set* problem, which is *NP*-complete [24]. In PACKER a greedy heuristic has been chosen to solve the problem. In each iteration of a loop the segment of maximal weight is taken from PA. The layer propagation routine is called for it, moving the segments that become incompatible from PA to M. Eventually, PA becomes empty and the iteration stops.

10) Move all segments in M to the next scan-line position.

VII. Top Level Control

In general a single scan through the switchbox will not be enough to resolve all conflicts and all wire segments which could not be given a place will accumulate at a row or column next to the switchbox boundary. A scan in another direction will have to solve the remaining conflicts.

Scanning can be done in any of the following directions: top–bottom (TB), right–left (RL), bottom–top (BT), and left–right (LR). Calling the scanning routine in this sequence or any of its permutations has the following property: if the layout is not modified (i.e., no segment is split, moved, or assigned another layer) during the four calls, then there are no conflicts left and a solution has been found. (A single scan ignores conflicts in the perpendicular segments and does not inspect the last row or column.) A sequence of calls of the four different scanning routines will be called one *iteration*.

In early stages of experimentation it turned out that the same sequence of calls in each iteration sometimes gave rise to getting trapped in a loop. For this reason PACKER uses a more sophisticated iteration scheme: in each iteration the sequence of calls is shifted cyclically with respect to the previous one. This iteration scheme is illustrated in Table I.

TABLE I
The Iteration Scheme.

Iteration no.	Sequence						
i	TB	RL	BT	LR			
$i+1$		RL	BT	LR	TB		
$i+2$			BT	LR	TB	RL	
$i+3$				LR	TB	RL	BT

After generating the initial routing, PACKER performs a number of iterations on the problem using the iteration scheme of Table I. As soon as an iteration (of four scans) does not modify the configuration, the algorithm stops, reporting success. At that moment some simple postprocessing is performed by removing most of the unnecessary detours in the wiring and most of the redundant vias. When no solution is found within the number of iterations given by the user, the algorithm stops and reports failure.

VIII. Neighbor Relations Constraining Segment Movement

The *neighbor relations* that will be discussed here are types of *vertical constraints* [25]. However, they are created and removed dynamically while the algorithm is in progress. A constrained segment can only move when a separation of at least one grid unit is maintained with other segments with which it has a neighbor relation.

Neighbor relations are created after solving a terminal conflict in the way of Fig. 9(b). There are two types of neighbor relations:

1) *higher* neighbors;
2) *lower* neighbors.

If segment A is a higher neighbor of B, B automatically is a lower neighbor of A. The interpretation of this relation is that A always should be on a grid line with a coordinate higher than the coordinate of the grid line of B. The following notation will be used: $B \mapsto A$, meaning that B is a lower neighbor of A. In the case of Fig. 9(b) the following four relations are created: $A \mapsto C$, $A \mapsto D$, $B \mapsto C$, and $B \mapsto D$. The neighbor relations are used to reduce the chance that a terminal conflict, which has been solved once, will reappear later on. Because a segment can be connected to terminal segments at both end points, it can have neighbor relations related to both terminals. In this way a complicated network of neighbor relations can exist.

A terminal conflict is only solved in the way described here, if the solution does not violate the existing constraints. Otherwise splitting is used as in Fig. 9(c). In this way no circular constraints are created; it cannot happen that the critical path of neighbor relations is longer than the number of rows or columns available. Neighbor relations can exist both between horizontal as well as between vertical segments.

Moving a segment that has constraint relations is a recursive action. Suppose segment s has to move one grid position in the direction of increasing coordinates. If there

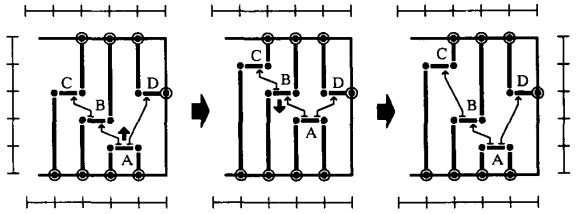


Fig. 12. Examples of moving segments with constraints. (a) Starting configuration. (b) After moving A upwards. (c) After moving B downwards.

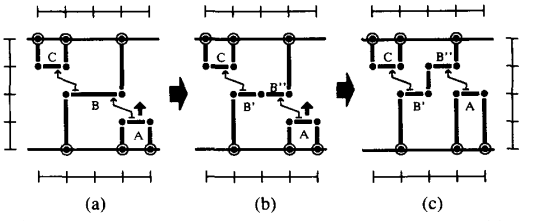


Fig. 13. Segment A in situation (a) cannot move to a higher position because of constraints; after splitting segment B as in situation (b) segment A can be moved resulting in situation (c).

is a segment t, such that:

- $s \mapsto t$;
- s and t are only separated by one grid unit.

t will also have to move in order to preserve the neighbor relations. Moving t is done in the same way as s. For this reason moving is recursive. Of course, moving in the direction of decreasing coordinates is an analogous action. An example of moving segments with the preservation of constraints is given in Fig. 12.

It is not always possible to move segments while preserving constraints; e.g., when the last segment in a chain of neighbor relations is next to the boundary, a move would push this segment on the boundary, which is an illegal position. For example, in Fig. 12(b) neither A nor B can move to a higher position. Not only the boundary can block a move; a neighbor relation with a terminal segment also can lead to blocking. The relation $A \mapsto D$ in Fig. 12 is an example of this case.

Although PACKER tries to avoid it, it can happen that a segment has to move, even when the neighbor relations disallow it. In such a case constraints should be broken. This can be done in many ways: each of the constraints in a chain of neighbor relations is a candidate for being removed. After removal of the constraint, performing the intended move will create a terminal conflict. However, the algorithm can continue its search without getting stuck.

Apart from simply removing constraints, splitting a segment in the chain can also be effective. Splitting helps because multiple constraints working on a segment are distributed over the two new segments (see Fig. 13). PACKER uses the following strategy when constraints prevent a move: it tries to split segments starting with those which are farthest away and going backwards to the segment that is intended to be moved. If this fails, it inspects the neighbor relations of the segment to be moved and removes those that prevent the move. For example, supposing that in Fig. 13 segment B had length 1, and therefore, could not be split, the algorithm would remove the relation $A \mapsto B$ and move A, making it overlap with B.

IX. Weight Functions

As explained in Section VI a weight is assigned to a segment to express the desirability of keeping the segment at the scan-line position. The computation of the weight is simple: a number of *weight functions* are called for the segment s for which the weight needs to be computed; each of them returns a positive, real number; the addition of these numbers determines the weight $w(s)$:

$$w(s) = \sum_i w_i(s), \qquad i = L, C, M, A, S.$$

The functions are the following.

- *Length Weight:* This weight increases linearly with the length of the segment: $w_L(s) = f_L * \text{length}(s)$. In a certain way this weight contributes to a higher packing density: if there are two segments competing to remain at the scan-line position, the longer one will be favored, resulting in a higher density.
- *Constraint Weight:* This is a two-valued function: $w_C(s) = W_C$ or 0. A value W_c is given when the wire cannot move because of the neighbor relations (see Section VIII), a value zero is given in other cases. This is how PACKER tries to avoid breaking constraints.
- *Merge Weight:* This is also a two-valued function: $w_M(s) = W_M$ or 0. The value zero is given when the segment has the possibility to merge with a segment of the same net further ahead (not necessarily at the next grid position) in the direction of the movement. Otherwise the value is W_M given. This weight helps to get rid of degenerated shapes like the one in Fig. 5(a).
- *Align Weight:* This is a three-valued function that inspects the end points of the segment: $w_A(s) = W_A$ or $1/2\ W_A$ or 0. If both end points are only connected to perpendicular segments, the weight is zero. For each end point that extends on the same line into another segment the value $1/2\ W_A$ is added. This weight helps to avoid too much jogging.
- *Selector Weight:* This weight function depends on the history. It is two-valued: $w_s(s) = W_s$ or 0. It is used to introduce disturbances in the algorithm, in order to avoid that it gets trapped in a loop. The algorithm has a variable, called **selector,** which is incremented by one modulo n, the number of nets in the switchbox, each time the scan line is moved. If the segment s belongs to the net with the net number equal to **selector,** an extra weight W_S is added to it. At the end of each "iteration" **selector** is corrected

by adding to it the smallest integer c such that $t + c$ and n are relatively prime, where t is the total number of scan-line positions in one iteration.

Summarizing, it can be said that there are a number of parameters that influence the weight: f_L, W_C, W_M, W_A, and W_S. However, this does not mean that each individual problem can only be solved with a tailored set of parameter values. The motivation for working with a "fixed" set of values together with some numerical data are given in Section XII.

X. Extra Splitting

The algorithm, containing the elements explained up to now, already proved rather successful. However, for some problems it happened that a long segment of a certain net permanently was swept up and down without finding a position where it could remain: it always was in conflict with a terminal segment. As a solution to this problem, it was decided to introduce extra splits in the segments. This has been done using the following method.

- The parameter **selector** (see Section IX) is also used here; it is used to select which of the segments are a candidate to split. In order not to interfere with its original function, **selector** is used here to select another net than the one which will receive an extra weight (e.g., with a net number which is one higher).
- Immediately after the terminals have been removed from PA in the algorithm of Section VI, those segments belonging to the net selected by **selector** and whose lengths are greater than or equal to the parameter $s_{L,\min}$ (the minimal split length), are split in two segments of equal length (when the original segment has an odd length, the new segments differ one unit in length).

In this way, splitting is performed at a limited rate, just often enough to avoid getting trapped in a loop. As will be explained in Section XII, the choice for the value of $s_{L,\min}$ turns out to be problem dependent.

XI. Worst-Case Time Complexity

The analysis of PACKER's worst-case time complexity uses the following three parameters:

- r, the size of the switchbox, the maximum of its height and width;
- n, the number of nets in the switchbox;
- s, the number of scans performed in the execution of the algorithm.

First the maximal number of segments that a net can have will be considered. In the worst case the route for the net will meander through the complete switchbox, using only unit length segments (see Fig. 14 for an example). Because overlapping of segments within the same net is not possible, the maximal number of segments per net is $\mathcal{O}(r^2)$. The number of segments crossed or covered by a scan line is $\mathcal{O}(r)$. Considering all the nets at the same time, the total number of segments is $\mathcal{O}(nr^2)$ and the number of segments crossed or covered by a scan line is $\mathcal{O}(nr)$.

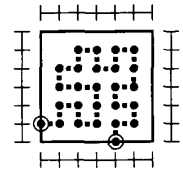


Fig. 14. A possible worst-case routing of a net.

The part of the algorithm that determines the overall time complexity is the "packing of segments," the actions performed for one scan-line position. Because there are a fixed number of layers, viz. 2, the number of segments selected to remain is $\mathcal{O}(r)$. Each of the segments selected to remain is found after a linear search through a list of $\mathcal{O}(nr)$ segments. So the selection process requires $\mathcal{O}(nr^2)$ of time. $\mathcal{O}(nr)$ segments will have to move to the next position. For each segment to be moved, the data base of segments are inspected to find those segments with which it has a grid point in common. Using the data structures of Section III, this requires $\mathcal{O}(\log^2 N)$ of time [20], if N is the total number of segments in the layout. So, moving requires $\mathcal{O}(nr \log^2 nr^2)$ of time. Taking selection and moving together gives $\mathcal{O}(nr^2 + nr \log^2 nr^2)$ of time for one scan-line position.

There are $\mathcal{O}(r)$ positions per scan and s scans giving a total time complexity of $\mathcal{O}(snr^3 + snr^2 \log^2 nr^2)$.

This expression can be simplified, when one realizes that the number of nets is bounded by the perimeter of the switchbox. Therefore n is $\mathcal{O}(r)$ and the complexity of PACKER becomes $\mathcal{O}(sr^4)$. So, in the worst case the computation time per scan is proportional to the square of the routing area.

XII. Tests And Results

The results achieved by PACKER are discussed in this section. First the measurement conditions (parameter settings, etc.) are given, together with a short motivation. Then the results obtained for the main "benchmarks" on switchbox routing are presented, followed by some results on channel routing. Finally, some attention is paid to the sensitivity of the algorithm to the "orientation" and to the "initial routing."

PACKER has been implemented in Common Lisp. All the timing results given refer to compiled code on an Apollo DN 4000 workstation with 8Mbytes of main memory.

12.1. Measurement Conditions

This section will outline the conditions under which the results, to be presented later on, have been obtained.

Two ways of generating the initial routing have been mentioned in Section IV. The method of the *quasi-minimal Steiner tree* has been used for all experiments, with

the exception of those where the intention was to compare the two ways of initial routing.

PACKER has a number of parameters that influence the weight functions (see Section IX) and the parameter $s_{L\min}$ (see Section X). It would be too much trouble for the user of PACKER to have to choose an appropriate set of parameter values for each problem to be solved. Instead, it has turned out that good results can be achieved when the user is given control of a single parameter: $s_{L\min}$.

The functions for constraint, merge, align, and selector weights (characterized by W_C, W_M, W_A, and W_S) do not take the length of a segment into account, whereas the length weight is proportional to the length. If W_C, W_M, W_A, and W_S would be kept constant for all problems, the contribution of the length weight would be larger in problems where the segment length is longer in average. In relation with this point, it should be noted that after a certain number of iterations most of the segments in the switchbox will have a length shorter than $s_{L\min}$. So, it is interesting to link the values of W_C, W_M, W_A, and W_S to the value of $s_{L\min}$. This has been done as follows:

$$W_C = s_{L,\min} * f_C$$

$$W_M = s_{L,\min} * f_M$$

$$W_A = s_{L,\min} * f_A$$

$$W_S = s_{L,\min} * f_S.$$

The parameter f_L associated with the length weight, gets the value 1.0 for purposes of normalization. f_C, f_M, f_A, and f_S which have been introduced above, are considered to be constants (all the experiments reported have been performed with $f_C = 1.0, f_M = 1.5, f_A = 0.8$, and $f_S = 1.0$). As far as the choice for $s_{L\min}$ is concerned, it turns out that the best performance of the algorithm is achieved when $s_{L\min}$ is set to about half of the maximal dimension of the switchbox: a higher value increases the probability that the algorithm will get trapped in a loop, whereas a lower value will increase the computation time because of the higher number of segments that have to be moved. For some problems this general rule does not apply: Fig. 15 shows the "thread" switchbox [8], that PACKER is only able to solve when $s_{L\min}$ is set to the value of 2. The unique solution requires the "staircase" pattern for the routing of net #6. For this reason, the solution can only be found when the route for net #6 consists of unit length segments, which, on its turn, is achieved when all segments longer than unit length have been split sooner or later.

12.2 Results for Well-Known Switchbox Routing Problems

In this section the results obtained by PACKER for a number of well-known switchbox routing problems will be presented and compared with the results of other successful routers. The paper on BEAVER [8] gives a well-documented overview of results obtained by a large variety of routers. It does not make much sense to reproduce the complete overview here. Instead, the comparison will be limited to the three most-successful routers in the overview: BEAVER, WEAVER [5], [26], and MIGHTY [10]; besides, two routers that are newer than [8]; SILK [12], and CODAR [13] will be taken into account. For the examples the same names will be used as in [8], except for "comp-1", which is a new example introduced in [13].

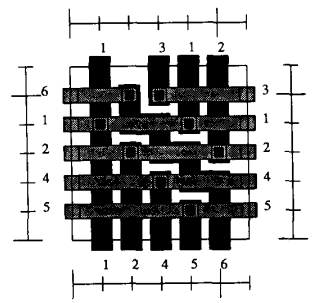


Fig. 15. PACKER's result for the "thread" switchbox.

TABLE II
PACKER'S RESULTS FOR SOME BENCHMARK SWITCHBOXES.

example's name	$s_{L,min}$	iterations	run time *sec*	modifications	wire length	vias
difficult	10	14	56	1041	546	45
more difficult	10	70	1400	24339	541	43
terminal intensive	10	28	210	3393	626	50
augmented dense	10	5	31	414	529	32
modified dense	10	6	36	391	510	29
pedagogical	8	14	91	1790	406	45
mixed pedagogical	6	17	75	1545	406	31
comp-1	10	14	160	3161	525	44

The reader should note that the results presented here are not the best results ever found by PACKER, but those obtained with the parameter choice explained earlier. Table II gives the relevant data for each example. These are, respectively, the value of the parameter $s_{L\min}$, the number of iterations performed to find the solution, the execution time, the number of modifications (segment moves, splits, and layer changes) performed to reach the solution from the initial routing, the total wire length, and the number of vias. The result for the "more difficult" switchbox is illustrated in Fig. 16. For most examples the wire length is somewhat shorter than the length reported in an earlier publication on PACKER [27] (and also the number of vias is often smaller). This is solely the merit of improved postprocessing; the main algorithm has not been modified.

The results are compared with other routers in Table III. The runtime has been included in the table in order to have a global idea of the performance. More cannot be done when different programming languages on different types of computers are used. In the case of PACKER, the goal was to build an experimental tool, which should be flexible rather than efficient.

As far as the comparison with BEAVER is concerned, the following should be remarked: BEAVER takes the freedom to assign a terminal to the most convenient layer with the goal of saving vias, whereas most routers assume

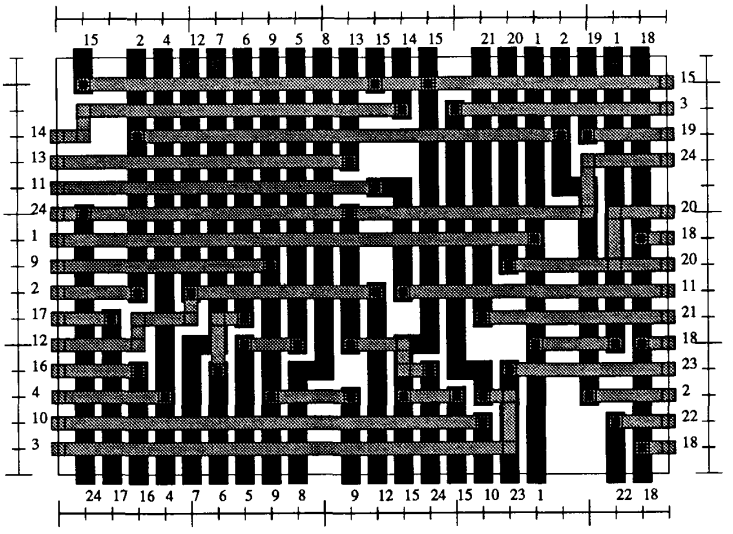


Fig. 16. A solution found by PACKER for the more difficult switchbox.

that terminals have a fixed layer. The last assumption is justified by the fact that in IC design, switchboxes abut to cells whose layout has already been designed: this means that the layer of a terminal is fixed by the specification. However, it turns out that all solutions produced by BEAVER can be converted to solutions obeying the original specifications of the benchmark examples by the introduction of extra vias. In the tables the number of vias for BEAVER has been corrected by incrementing the original number with this number of extra vias. The original number is given inside the parentheses.

The fixed-layer restriction does not mean that PACKER requires that all terminals along one edge have the same layer, even when the benchmark examples have this property. To illustrate this case, PACKER has been run on an example called the *mixed pedagogical switchbox*. It uses the layer assignment for terminals produced by BEAVER in its solution to the *pedagogical switchbox*, an example introduced by BEAVER itself. An illustration of the result is given in Fig. 17.

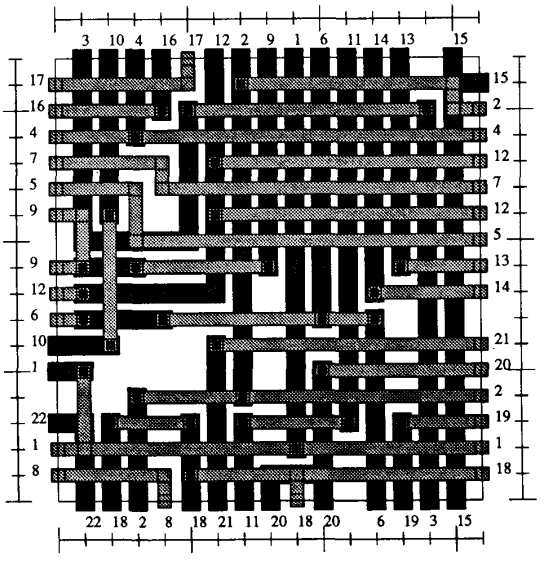


Fig. 17. A solution found by PACKER for the mixed pedagogical switchbox.

12.3. Some Results for Channel Routing

Channel routing problems that do not have floating terminals at the "open sides" can be described as switchbox routing problems, when the number of rows is fixed in advance. PACKER can solve small and medium-size channel routing problems, that have the mentioned property. Big problems, such as the *Deutsch difficult channel* [28] are out of the scope of PACKER because of the nonlinear time complexity (see Section XI).

One of the most interesting results achieved by PACKER is a solution to a problem originally published in [29]. A 5-row solution required two empty rows in the middle using a routing model where each layer had its own orientation for wire segments. MIGHTY, without the restricted routing model, solved the problem using 4 rows and a single empty column in the middle. PACKER, using the same unrestricted model, is able to solve it in 4 rows, without any empty columns. The solution is given in Fig. 18.

Two other channels, which are not very easy to solve, will be presented here: they originate from Joobbani's thesis on WEAVER [26] and will be called here *joo6-12* and *joo6-13* after the figure numbering in the thesis. WEAVER solves *joo6-13* in 7 rows; SILK [12] succeeds in solving it using 6 rows; PACKER is able to do the same (see Fig. 19; the problem with six rows has been called *joo6-13-6r*). The data for the channel routing examples

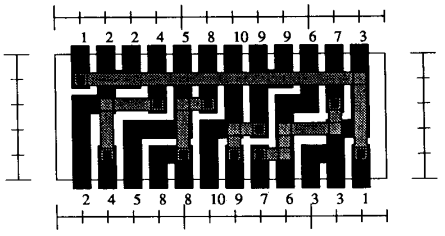


Fig. 18. A solution found by PACKER for Burstein's difficult channel.

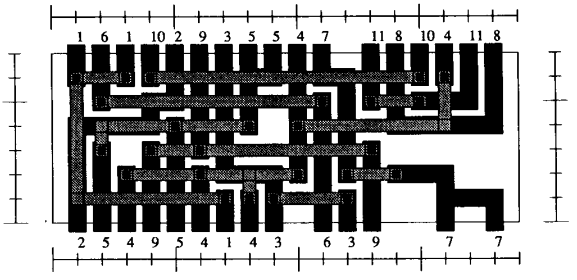


Fig. 19. A solution found by PACKER for the channel *joo6-13-6r*.

TABLE III
COMPARISON OF PACKER WITH OTHER ROUTERS FOR WELL-KNOWN SWITCHBOXES.

example	router	wire length	vias	run time sec
difficult	WEAVER	531	41	1500
	BEAVER	547	43 (35)	1
	CODAR	544	?	15
	PACKER	546	45	56
more difficult	MIGHTY	541	39	4
	BEAVER	536	42 (34)	1
	SILK	528	36	69
	CODAR	545	?	17
	PACKER	541	43	1400
terminal intensive	WEAVER	615	49	1800
	MIGHTY	629	50	?
	BEAVER	632	53 (46)	1
	SILK	616	49	?
	CODAR	630	?	21
	PACKER	626	50	210
augmented dense	MIGHTY	530	32	?
	BEAVER	529	32 (27)	1
	CODAR	529	?	10
	PACKER	529	32	31
modified dense	WEAVER	510	29	920
	MIGHTY	510	29	?
	BEAVER	510	29 (26)	1
	SILK	510	29	?
	CODAR	510	?	11
	PACKER	510	29	36
pedagogical	BEAVER	396	38 (31)	1
	PACKER	406	45	91
mixed pedagogical	BEAVER	396	31	1
	PACKER	406	31	75
comp-1	MIGTHY	fail	-	-
	CODAR	529	42	50
	PACKER	525	44	160

are given in Table IV, whereas the comparison with WEAVER and SILK is made in Table 5.

12.4. Sensitivity for Orientation

Given the top level control structure with a fixed sequence of scans (see Section VII), it should be clear that

TABLE IV
PACKER'S RESULTS FOR SOME CHANNEL ROUTING PROBLEMS.

example's name	$s_{L,min}$	iterations	run time sec	modifications	wire length	vias
difficult channel	3	38	87	2017	82	10
joo6-12	2	6	5.6	132	82	18
joo6-13	5	9	31	703	185	26
joo6-13-6r	5	143	710	18674	167	25

TABLE V
COMPARISON OF PACKER WITH OTHER ROUTERS FOR SOME CHANNNELS.

example	router	wire length	vias	run time sec
joo6-12	WEAVER	82	14	?
	PACKER	82	18	5.6
joo6-13	WEAVER	167	29	310
	PACKER	185	26	31
joo6-13-6r	SILK	166	29	?
	PACKER	167	25	710

PACKER's behavior is not invariant for different orientations (rotations, mirrorings) of the same problem. In order to investigate this point, PACKER was run on the 8 possible orientations of Burstein's difficult switchbox. The results are shown in Table VI. Note that some of the results are qualitatively better than the "regular" result for the example. When the runtimes are compared, it can be seen that the fastest solution requires about 15 times less CPU time than the slowest one. Considering the nature of PACKER, this is not a surprise: when different paths in the space of configurations are followed, the moment at which the path will end in a solution, is subject to chance. The fact that a solution was found in all cases, shows however, the robustness of the algorithm.

12.5. Sensitivity for Initial Routing

As has been explained in Section IV, PACKER incorporates two methods for the generation of the initial routing: *quasi-minimal Steiner tree* and *counterclockwise routing along the boundary*. Intuitively, the first method generates a configuration closer to a solution than the configuration produced by the second one. This intuitive notion has been tested for some of the examples mentioned earlier by trying to route them starting with a configuration generated by the counterclockwise routing. The results are shown in Table VII. It is important to note that the algorithm did not fail for any of the examples, although in some cases a different value for $s_{L,\min}$ had to be chosen. This is an additional indication for the robustness of the algorithm.

In Table VIII the two methods for the generation of the initial routing are compared. The table gives the wire length after the initial routing, the final wire length, and the execution time for both methods. The initial wire length is included in the table to give a rough idea of the distance between the initial configuration and the final solution. From the table it can be concluded that in a few cases counterclockwise routing leads to a faster solution, but that it can also slow down the time necessary to find a solution by a factor of 40.

TABLE VI
PACKER'S RESULTS FOR ALL ORIENTATIONS OF BURSTEIN'S DIFFICULT EXAMPLE.

example's name	$s_{L,min}$	iterations	run time sec	modifications	wire length	vias
difficult	10	14	56	1041	546	45
90 deg. rotation	10	9	28	418	562	50
180 deg. rotation	10	19	290	4002	546	46
270 deg. rotation	10	20	220	4459	552	47
x-axis mirroring	10	33	410	8092	569	41
y-axis mirroring	10	48	92	1551	536	42
y=x mirroring	10	17	74	1302	542	43
y=-x mirroring	10	32	320	6869	548	48

TABLE VII
PACKER'S RESULTS USING THE COUNTERCLOCKWISE INITIAL ROUTING METHOD.

example's name	$s_{L,min}$	iterations	run time sec	modifications	wire length	vias
difficult	9	51	650	12700	559	44
more difficult	10	65	910	15183	548	42
terminal intensive	9	60	1200	17550	620	51
augmented dense	10	184	1300	14881	529	31
modified dense	10	5	160	1549	510	29
pedagogical	8	438	4300	84855	414	31
comp-1	10	8	140	2045	523	40
difficult channel	2	10	20	394	82	18
joo6-12	2	10	27	654	80	14
joo6-13	5	27	130	3103	182	29
joo6-13-6r	6	920	4100	112595	175	25

TABLE VIII
COMPARISON OF THE TWO WAYS OF INITIAL ROUTING IN PACKER.

	quasi-Steiner			counterclockwise		
example's name	initial length	final length	run time sec	initial length	final length	run time sec
difficult	526	546	56	976	559	650
more difficult	514	541	1400	940	548	910
terminal intensive	601	626	210	1063	620	1200
augmented dense	536	529	31	842	529	1300
modified dense	517	510	36	825	510	160
pedagogical	389	406	91	543	414	4300
comp-1	496	525	160	909	523	140
difficult channel	82	82	87	166	82	20
joo6-12	77	82	5.6	133	80	27
joo6-13	160	185	31	283	182	130
joo6-13-6r	152	167	710	275	175	4100

XIII. CONCLUSION

The results show that PACKER can compete very well with other switchbox routers, especially as far as the degree of successful solutions is concerned. Below the most significant points of PACKER are summarized.

- *The routing process is directly guided by information on conflicts:* The fact that all nets are always connected, makes the locations with conflicts directly available. In this way the interaction of the nets is dealt with in a straightforward way.
- *Reshaping is the only way of modification:* PACKER has shown that the the use of connectivity preserving local transformations as the only way to modify a layout (as opposed to "rip-up and reroute," which is much more an action with global impact) is a viable alternative to routing.
- *The combination of "scanning" and "packing" is a powerful technique for finding a solution:* The danger of an algorithm based on local transformations is that it will get stuck in a local minimum, because of a limited horizon. The scanning technique of PACKER is able to move segments from one side of the switchbox to the other one. In this way it "reshuffles" the nets considerably, such that local minima are avoided. On the other hand, "packing" of segments at a single scan-line position is an appropriate method for the efficient utilization of the routing area. Actually, it is due to packing that the algorithm converges to a solution.

Recently, the principles of PACKER have been extended to be used in a *general area* router, called CRACKER [30]. It successfully handles routing problems with irregular boundaries, floating terminals, terminals inside the routing area, and obstacles in either of the two layers.

ACKNOWLEDGMENT

The authors would like to thank Prof. Jess from Eindhoven University and Prof. Otten from Delft University for their valuable comments on the topic. Many corrections of the text were suggested by Sonia Heemstra who always was prepared to read subsequent versions of this article.

REFERENCES

[1] J. Soukup, "Circuit layout," *Proc. IEEE*, vol. 69, pp. 1281-1304, Oct. 1981.
[2] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. Elect. Comput.*, vol. EC-10, pp. 346-365, Sept. 1961.
[3] T. Ohtsuki, "Maze-running and line-search algorithms," in T. Ohtsuki, Ed. *Advances for CAD in VLSI, Vol. 4: Layout Design and Verification*. Amsterdam, The Netherlands: North-Holland, 1986, pp. 99-131.
[4] W. P.-C. Ho, D. Y. Y. Yun, and Y. H. Hu, "Planning strategies for switchbox routing," in *Proc. Int. Conf. Computer Design*, pp. 463-467, 1985.
[5] R. Joobbani and D. Siewiorek, "Weaver: A knowledge-based routing expert," *IEEE Design Test*, vol. 3, pp. 12-23, Feb. 1986.
[6] R. L. Rivest and C. M. Fiduccia, "A greedy channel router," in *Proc. 19th Design Automation Conf.*, pp. 418-424, 1982.
[7] S. Chen, M. Lin, and W. Zhuang, "A new algorithm for four-side channel routing," in *Proc. China Int. Conf. Circuits and Systems*, pp. 73-76, 1985.
[8] J. P. Cohoon and P. L. Heck, "Beaver: A computational-geometry-based tool for switchbox routing," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 684-697, June 1988.
[9] W. K. Luk, "A greedy switch-box router," *Integration, The VLSI J.*, vol. 3, pp. 129-149, 1985.
[10] H. Shin and A. Sangiovanni-Vincentelli, "A detailed router based on incremental routing modifications: Mighty," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 942-955, Nov. 1987.
[11] C. S. Ying, X. L. Hong, and E. Q. Wang, "Draft: An efficient area router based on global analysis," in *Proc. Conf. Computer-Aided Design*, pp. 386-389, 1987.
[12] Y. L. Lin, Y. C. Hsu, and F. S. Tsai, "A detailed router based on simulated evolution," in *Proc. Int. Conf. Computer-Aided Design*, pp. 38-41, 1988.
[13] P. S. Tzeng and C. H. Séquin, "Codar: A congestion-directed general area router," in *Proc. Int. Conf. Computer-Aided Design*, pp. 30-33, 1988.
[14] F. Rubin, "An iterative technique for printed wire routing," in *Proc. 11th Design Automation Workshop*, pp. 308-313, 1974.
[15] E. Rosenberg, "A new iterative supply/demand router with rip-up capability for printed circuit boards," in *Proc. 24th ACM/IEEE Design Automation Conf.*, pp. 721-726, 1987.

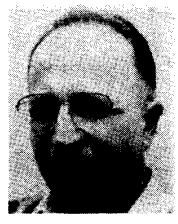
[16] R. Linsker, "An iterative-improvement penalty-function driven wire routing system," *IBM J. Res. Develop.*, vol. 28, no. 5, pp. 613–624, Sept. 1984.
[17] K. A. Chen, M. Feuer, K. H. Khokhani, N. Nan, and S. Schmidt, "The chip layout problem: An automatic wiring procedure," in *Proc. 14th Design Automation Conf.*, pp. 298–301, 1977.
[18] R. S. Fisher, "A multi-pass, multi-algorithm approach to PCB routing," in *Proc. 15th Design Automation Conf.*, pp. 82–91, 1978.
[19] E. M. McCreight, "Priority search trees," *SIAM J. Comput.*, vol. 14, no. 2, pp. 257–276, May 1985.
[20] T. Asano, M. Sato, and T. Ohtsuki, "Computational geometry algorithms," in T. Ohtsuki, Ed., *Advances in CAD for VLSI, Vol. 4: Layout Design and Verification.* Amsterdam, The Netherlands: North-Holland, 1986, pp. 295–347.
[21] R. C. Prim, "Shortest connection networks and some generalizations," *Bell Syst. Tech. J.*, vol. 36, pp. 1389–1401, Nov. 1957.
[22] M. W. Bern, "Two probabilistic results on rectilinear Steiner trees," in *Proc. 18th Ann. Symp. Theory Computing*, pp. 433–441, 1986.
[23] J. P. Cohoon, D. S. Richards, and J. S. Salowe, "A linear-time Steiner tree routing algorithm for terminals on the boundary of a rectangle," in *Proc. Int. Conf. Computer-Aided Design*, pp. 402–405, 1988.
[24] A. Gibbons, *Algorithmic Graph Theory.* Cambridge, U.K.: Cambridge University, 1985.
[25] B. W. Kernighan, D. G. Schweikert, and G. Persky, "An optimum channel-routing algorithm for polycell layouts of integrated circuits," in *Proc. 10th Design Automation Workshop*, pp. 50–59, 1973.
[26] R. Joobbani, "WEAVER: An application of knowledge-based expert systems to detailed routing of VLSI chips," Ph.D. dissertation, Dep. Elec. Comput. Eng., Carnegie Mellon Univ., June 1985.
[27] S. H. Gerez and O. E. Herrmann, "Packer: A switchbox router based on conflict elimination by local transformations," in *Proc. Int. Symp. Circuits Systems*, pp. 961–964, 1989.
[28] D. N. Deutsch, "A dogleg channel router," in *Proc. 13th Design Automation Conf.*, pp. 425–433, 1976.
[29] M. Burstein and R. Pelavin, "Hierarchical channel router," *Integration, The VLSI J.*, vol. 1, pp. 21–38, 1983.
[30] S. H. Gerez and O. E. Herrmann, "Cracker: A general area router based on stepwise reshaping," in *Proc. Conf. Computer-Aided Design*, 1989.

*

Sabih H. Gerez received the M.Sc. degree in electrical engineering from the University of Twente, The Netherlands, in 1984. He is currently working towards the Ph.D. degree at the University of Twente.

He has worked as assistant researcher in the group for network theory and VLSI design at the Faculty of Electrical Engineering, University of Twente. His research interests include computer-aided design for VLSI, with an emphasis on automatic layout generation, the design of computer algorithms in general, and formal languages for programming and hardware description.

*



Otto E. Hermann (M'72) received the Dipl.-Ing. and the Dr.-Ing. degrees in electrical engineering from the University of Technology, Aachen, Germany, in 1959 and 1965, respectively.

From 1959 to 1965, he was a research associate and lecturer at the University of Aachen and the University of Karlsruhe. From 1966 to 1972, he was a senior staff member, and from 1972 to 1975, he was "Abteilungsvorsteher" for the Department of Electrical Engineering, University of Erlangen. Since 1975, he has been a Full Professor on the Faculty of Electrical Engineering, University of Twente, The Netherlands, heading the professional group for network theory, signal processing, and CACSD.