CRACKER: A General Area Router Based on Stepwise Reshaping

Sabih H. Gerez and Otto E. Herrmann

University of Twente, Faculty of Electrical Engineering
Laboratory for Network Theory
P.O.B. 217, 7500 AE Enschede, The Netherlands

Abstract

CRACKER is an algorithm able to handle a large class of routing problems. Operating on a grid and using two wiring layers, it can deal with floating and fixed terminals, arbitrarily located in the routing area, and with obstacles in either of the two layers. The routing process consists of two stages: in the first stage, all nets are interconnected quickly, without avoiding conflicts with previously routed nets or obstacles; in the iterative second stage, *connectivity preserving local transformations* are applied in a systematic way, such that, eventually, a solution without conflicts is reached. There is no rip-up and reroute. The same algorithm is able to solve well-known examples of switchbox routing, routing with irregular boundaries, L-shaped channels, three-sided channels, etc.

1 Introduction

This paper explains how the principle of *stepwise reshaping* can be applied to general area routing problems, as implemented in the CRACKER algorithm. When using this principle, all nets are first connected quickly and independently of each other at the expense of conflicts (short circuits), and then a solution is searched for iteratively. In the iterative phase the layout is solely modified by means of *connectivity preserving local transformations* until all conflicts have disappeared. Because all nets are always connected, the information on the locations of conflicts is directly available to guide the control of the transformations.

This technique has already been applied with success for switchbox routing in the PACKER algorithm [6,7]. It turns out that the technique is also applicable to less restricted routing problems, to *general area routing* [13].

The class of routing problems to which CRACKER can be applied, is given by the following routing model:

1. An orthogonal grid is used.
2. Two layers are available for routing.
3. Both layers can accommodate horizontal and vertical routing segments.
4. The specification of the problem either fixes the position of a terminal on one grid point, or allows the position to be one of a group of adjacent grid points delimited by a rectangle. The specification can fix the layer of a terminal, but does not need to do so.
5. Obstacles can occupy any of the grid points in either of the two layers.

The class of routing problems just described includes many problems for which special-purpose algorithms have been developed. Examples are: conventional switchbox routing [4], switchboxes with irregular boundaries [10], three-sided channels [10] and L-shaped channels [3].

The main difference between CRACKER and other routers that can deal with a large class of routing problems [9,12,13] is the fact that CRACKER relies on stepwise reshaping operations *only*. Although the "weak modifications" of [12] and the "wire pushing operations" of [13] are quite similar, they are used in combination with other methods of modification, especially with *rip-up and reroute*.

This paper is organized as follows: the algorithm is introduced in a bottom-up fashion and later the different results obtained are discussed.

2 The Representation of Routing and Obstacles

The routing of a net is represented by *nodes*, located on grid points, and *segments*, that interconnect the nodes. A *terminal node*, which represents a terminal, contains information on a rectangular area that covers all its legal positions. In practice the rectangle always degenerates into a point, for fixed terminals, or a line segment, for terminals floating along a boundary. The existing option of a terminal floating within a two dimensional area does not seem to have any useful application in the current CAD practice. Information on the layer is stored in a segment. A node connected to segments in different layers implicitly represents a contact cut. The segments are stored in a data structure for the easy detection of conflicts, which is based on the *priority search tree* [1,11].

A straightforward way to represent the obstacles is to use for them the same data structures as for routing: therefore obstacles are mapped on segments connected to a node at both end points (even when an obstacle occupies a single grid point). The obstacles need not be interconnected with each other, as they have a permanent position and are not reshaped.

This uniform way of representing routing segments and obstacles has many advantages. Conflicts of routing segments with each other and conflicts of routing segments with obstacle segments can be detected in the same way. Another point is that the router can operate inside a simple rectangular area, irrespective of the actual boundary: the obstacles modelling the shape of the boundary are contained in the rectangle.

3 Initial Routing

The initial routing procedure in CRACKER is based on a quasi-minimal rectilinear Steiner tree heuristic. The routine is executed for each net separately; neither the nets that have been laid out in

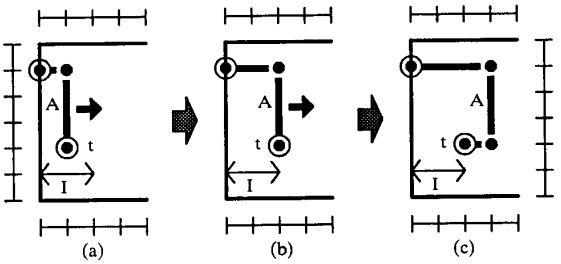

Figure 1: *Segment A is connected to a floating terminal node t which is allowed to float inside the interval I (a); when A is moved twice to the right, t changes its location the first time (b), but not the second time (c).*

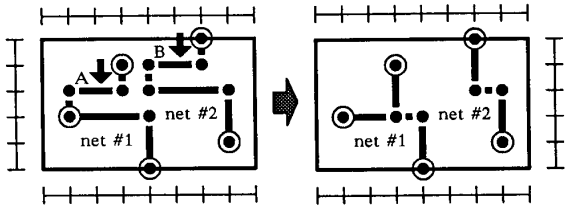

Figure 2: *Nets #1 and #2 have a similar structure, but net #1 has an extra terminal node. Therefore, the structures resulting after moving A and B differ.*

a previous stage, nor the obstacles influence the result, with the effect that many conflicts are likely to be created. More details on the Steiner tree heuristic can be found in [6,7].

4 Connectivity Preserving Local Transformations

The routing is modified by connectivity preserving local transformations (CPLTs in short). The three types of CPLTs, in increasing order of complexity, are:

1. *The layer change:* this simply involves a modification of the layer field of the segment data structure.
2. *The segment split:* any segment whose length is equal to or greater than 2 can be split into two segments with a node interconnecting them.
3. *The segment move:* this involves moving a segment one grid position aside, while it remains connected to the rest of the net.

Figures 1 and 2 illustrate some segment moves related to general area routing; other examples (related to switchbox routing, but also valid here) can be found in [7].

5 Scanning and Packing

The CPLTs have to be applied in a systematic way in order to enforce convergence towards a solution. In CRACKER this is done by a *scan line* technique: a scan line is positioned on subsequent grid lines and at each position as much as possible segments located under the scan line are packed in the two available layers, whereas those for which there is no space, are moved to the next position by means of CPLTs.

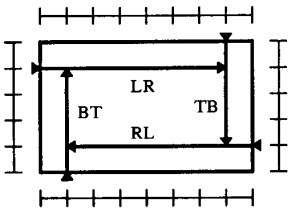

Figure 3: *The ranges of each scan.*

Scanning can be done in four directions: from top to bottom (TB), from bottom to top (BT), from left to right (LR) and from right to left (RL). A scan in a certain direction starts at a grid line coinciding with a boundary and stops at the grid line just before the opposite part of the boundary. All conflicts are accumulated on this opposite part of the boundary. A scan in another direction will have to solve the remaining conflicts. Figure 3 shows the ranges of the four scans.

The procedure executed for each scan line position, called the *packing* procedure, has the following steps:

1. Form PA, the set of parallel segments overlapped by the scan line. Also form PE, the set of perpendicular segments crossed by the scan line. Initialize M, the set of segments that will be moved, to the empty set.
2. Transfer the obstacle segments from PA and PE to O, the set of obstacles.
3. Inspect all segments in PA. If one of the layers at a segment's position is occupied by elements of O, the segment is moved to the opposite layer. If both layers are occupied, the segment is transferred from PA to M.
4. Remove from PA those segments that receive the *terminal status.* To say it in a simplified way, these are segments connected to non-floating terminal nodes, that don't conflict with each other (they already don't conflict with those in O). Add these segments to O.
5. Remove from PE those segments that are incompatible with segments in O, because they cross in the same layer. They will be ignored in the rest of the procedure.
6. Make compatible the segments in PA with those in PE and O, in a similar way as in Step 3. The incompatible segments are moved from PA to M.
7. Assign a *weight* to all the remaining segments in PA; the weight expresses the desirability to keep the segment at the current position.
8. Repeat until PA is empty: remove the segment with highest weight from PA and move those segments which become incompatible by this choice from PA to M.
9. Move all segments in M to the next scan line position by means of CPLTs.

Essentially, while packing the parallel segments the assumption is made that the perpendicular ones have a correct layer and position. The repeated use of this assumption, while changing the direction of scanning, leads CRACKER to convergence.

6 Top Level Control

The top level control of CRACKER consists of a number of *iterations*, each iteration calling the scanning routine for each of the four directions in some order. An iteration has the following

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

Table 1: *The iteration scheme of CRACKER.*

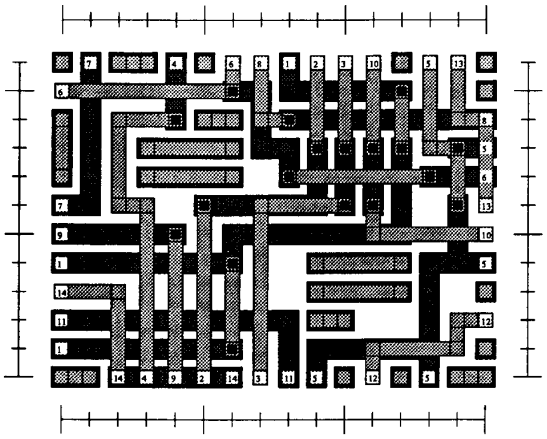

Figure 4: *A solution to a switchbox with obstacles; both the wiring and the obstacles have been displayed as represented internally.*

property: if the layout is not modified during the four calls, one is sure that there are no conflicts left and that a solution has been found. (A single scan ignores conflicts in the perpendicular segments and does not inspect the last row or column.)

The iteration scheme of CRACKER has been chosen in such a way that all possible subsequent pairs of different scans (12 in total) occur once in each three iterations, as can be seen in Table 1. The router executes this scheme until a solution is found or a maximum number of iterations is reached. When a solution is found, some post-processing is performed (simple via optimization and wire length reduction).

7 Results

This section presents different types of results obtained by CRACKER. CRACKER has been implemented in Common Lisp; the run time figures refer to compiled code on an Apollo 4000 workstation. Lisp was chosen as the language of implementation because it provides a nice environment for fast prototyping.

1. *Routing with obstacles:* The example presented in Figure 4 originates from [4]. It is a switchbox with two obstacles that block both layers in the middle of the routing area. The figure shows the obstacles together with the wiring. As in the original problem formulation, the layers of the terminals are left open in the specification. For CRACKER, this means that there are no single layer obstacles on the boundary forcing the terminals to be in the other layer; those parts of the boundary without terminals have been blocked by obstacles. Note that nets #13 and #14 have partially been routed *on* the boundary; when terminals can be in any of the two layers, both layers are also available for routing. CRACKER takes advantage of this situation.

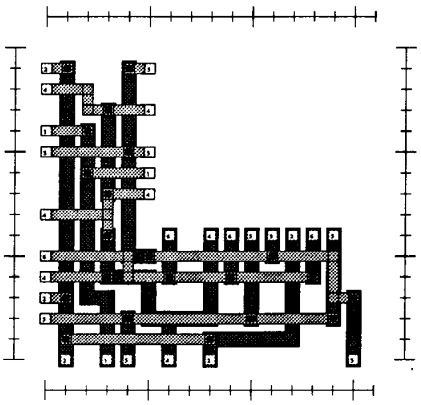

Figure 5: *CRACKER's solution for an L-shaped channel.*

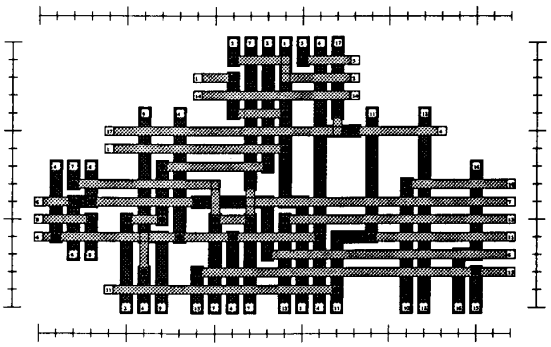

Figure 6: *CRACKER's solution for a switchbox with irregular boundaries.*

2. *L-shaped channels:* the example presented in Figure 5 originates from [3]. The L-shaped channel was modelled by putting obstacles in the upper right corner.

3. *Irregular boundaries:* here a problem from Liu et al. [10] is presented. They solved it by decomposing the routing area in small rectangular subproblems. CRACKER solves the same problem as a whole, see Figure 6.

4. *Three-sided channel:* the idea of taking Burstein's difficult switchbox [2] and making the bottom terminals floating was used in [10]. For CRACKER the following, more complicated, problem has been composed: all nets are given a floating terminal on the bottom, which requires an extra column, assigned at the right. The solution is displayed in Figure 7.

5. *Conventional switchbox routing:* Being a successor of the switchbox router PACKER [6,7], CRACKER is very suc-

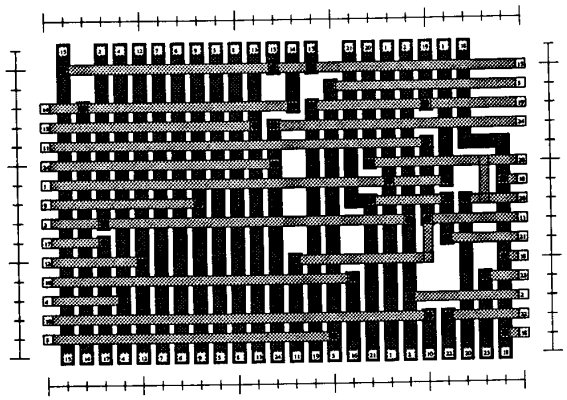

Figure 7: *CRACKER's result for a three-sided channel.*

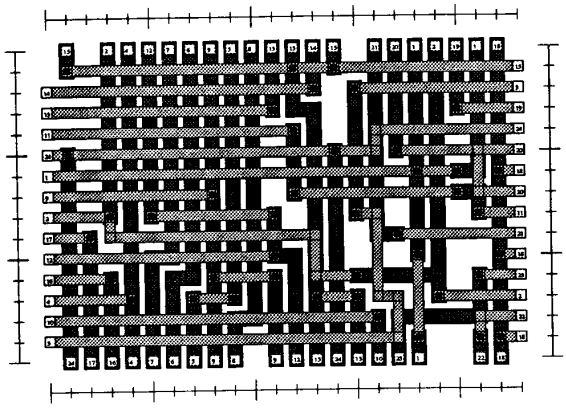

Figure 8: *A solution found by CRACKER for "comp-1".*

cessful for switchbox routing. It solves all the problems also solved by PACKER, including the "difficult", the "more difficult" and the "terminal intensive" switchboxes [4]. A solution for a compressed variant of the difficult example, "comp-1" [13], is given in Figure 8. It turns out that CRACKER is in general a factor 2 to 3 slower than PACKER. This can be explained by the fact that segments connected to terminal nodes cannot move in PACKER, they can only be split. In CRACKER any segment can move in principle; therefore, the search space is larger.

Data about the solutions mentioned above can be found in Table 2.

example	iterations	run time *sec*	modifications	wire length	vias
Figure 4	12	70	1905	180	19
Figure 5	17	110	1782	161	21
Figure 6	6	61	927	360	32
Figure 7	11	220	2511	605	36
Figure 8	13	220	4064	521	42

Table 2: *The results obtained by CRACKER for the examples shown in the figures.*

8 Final Remarks

The main conclusion is that CRACKER is able to solve many different types of routing problems, without using special-purpose techniques. More details on the algorithm and more examples can be found in [5].

It should be remarked that the CRACKER algorithm cannot easily be classified in a well-known category. As an algorithm that transforms a configuration by means of small steps into a neighboring configuration in the space of configurations, it has something in common with algorithms based on simulated annealing [8]. However, it is not possible to submit its convergence behavior to a thorough quantitative analysis, although, qualitatively, the "packing" strategy obviously helps to converge. At the same time, the "scanning" mechanism avoids getting trapped in local minima by sweeping segments all across the routing area. It turns out that CRACKER, after having walked rather randomly through configuration space, suddenly *locks in* to a solution. Although a behavior like this is perhaps undesirable, the rate of success of the algorithm makes it worth to study this type of algorithms seriously.

References

[1] T. Asano, M. Sato, and T. Ohtsuki. Computational geometry algorithms. In T. Ohtsuki, editor, *Advances in CAD for VLSI, Vol. 4: Layout Design and Verification*, pages 295–347, North-Holland, Amsterdam, 1986.

[2] M. Burstein and R. Pelavin. Hierarchical wire routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-2(4):223–234, 1983.

[3] H.H. Chen. Routing L-shaped channels in nonslicing-structure placement. In *24th ACM/IEEE Design Automation Conference*, pages 152–158, 1987.

[4] J.P. Cohoon and P.L. Heck. Beaver: a computational-geometry-based tool for switchbox routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(6):684–697, June 1988.

[5] S.H. Gerez. *Local Wire Routing by Stepwise Reshaping.* PhD thesis, University of Twente, Faculty of Electrical Engineering, December 1989.

[6] S.H. Gerez and O.E. Herrmann. Packer: a switchbox router based on conflict elimination by local transformations. In *International Symposium on Circuits and Systems*, pages 961–964, 1989.

[7] S.H. Gerez and O.E. Herrmann. Switchbox routing by stepwise reshaping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1989. Accepted for publication.

[8] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–690, May 1983.

[9] Y.L. Lin, Y.C. Hsu, and F.S. Tsai. A detailed router based on simulated evolution. In *International Conference on Computer-Aided Design*, pages 38–41, 1988.

[10] M.L. Liu, W.J. Zhuang, and S.Q. Chen. Three-side channel, two-adjacent-side channel and their routers. In *International Symposium on Circuits and Systems*, pages 1226–1229, 1986.

[11] E.M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, May 1985.

[12] H. Shin and A. Sangiovanni-Vincentelli. A detailed router based on incremental routing modifications: Mighty. *IEEE Transactions of Computer Aided Design of Integrated Circuits*, CAD-6(6):942–955, November 1987.

[13] P.S. Tzeng and C.H. Séquin. Codar: a congestion-directed general area router. In *International Conference on Computer-Aided Design*, pages 30–33, 1988.