An Algorithm for Embedding a Class of Non-Even Routing Problems in Even Routing Problems

Dee Parks

Mathematical Sciences
Appalachian State University
Boone, NC 38608

Miroslaw Truszczyński

Computer Science
University of Kentucky
Lexington, KY 40506

Abstract

In this paper we present part of a complete solution of a two-terminal net routing problem for certain non-convex grids without holes that we call channel graphs. We present an algorithm for embedding a non-even channel graph routing problem in an even channel graph routing problem. We refer to the algorithm as EMBED. EMBED *runs in time $O(b)$, where b is the number of vertices on the boundary, and is similar to an algorithm described by Becker and Mehlhorn [1] for planar graphs. Due to the restrictions we place on the shape of channel graph routing problems, however, we are able to obtain a lower complexity for our algorithm than that of the Becker and Mehlhorn algorithm. Their algorithm has complexity $O(bn)$, where n is the number of vertices in the graph.*

1 Introduction

In this paper we extend our work on Z-grids [5] to a class of 2-terminal net routing problems that we call *channel graph routing problems.* We have developed two algorithms, a routing algorithm and an embedding algorithm, which together form a complete solution of the channel graph routing problem. In this paper, we present only the embedding algorithm. The embedding algorithm converts a non-even instance of a channel graph routing problem to even, if it is possible to do so without oversaturating a cut.

Our work on embedding algorithms was undertaken as an attempt to find fast algorithms for use in routing general grid graphs. Lai and Sprague[4] describe an embedding algorithm for convex grid graphs that has complexity $O(b)$, but to our knowledge no fast embedding algorithms for more complex classes of graphs have been developed, other than those we describe here and in [5]. Kaufmann and Kläar[2] describe a $O(b \lg^2 b)$ embedding algorithm for non-convex graphs without rectilinear visible corners. An embedding algorithm for general non-convex graphs with rectilinear visible corners, described by Kaufmann and Mehlhorn[3], has complexity $O(n \lg^2 n + u^2)$, where u is the number of odd vertices on the boundary.

2 Definitions and Preliminaries

In order to define the class of graphs called channel graphs we require a definition of the term *rectangular subgraph.* A rectangular subgraph of a grid graph is a subgraph whose boundary edges form either a rectangle or a single horizontal sequence of edges.

Definition 1 (Channel Graph) *A grid-graph $G = (V, E)$ is a* channel graph *if it meets the following requirements:*

1. *G contains no holes,*
2. *there is a rectangular subgraph $C \subseteq G$ such that, from any vertex of G that is not in C, there is a straight path to some vertex in C, and*
3. *the vertices in $G - C$ form disjoint rectangular subgraphs.*

Definition 2 (Channel) *Given a channel graph G, we refer to the maximal rectangular subgraph of G from which there is a straight path to any other vertex in G as the* channel.

Not every grid graph without holes contains a channel. However, if a grid graph without holes does contain a channel, then any two vertices of the graph can be connected by a path having no more than two bends. Figure 1 shows two channel graphs, one in which the channel contains more than one horizontal sequence of edges, and one in which the channel is a single sequence of edges. We have explicitly shown the vertices in the channel to distinguish them from the vertices in the remainder of the graph.

We refer to the rectangular subgraphs of G that are not in the channel as *sideboxes* of G. Without loss of generality, we may assume that channel graphs are embedded in the plane so that all sideboxes are either above or below the channel. With this in mind, we classify sideboxes as either *top sideboxes* or *bottom sideboxes.* A sidebox may connect to the channel at two concave corners, or it may be flush with the left or right end of the channel. We place certain restrictions on the location of net terminals around the boundary of a channel graph.

0-8186-2610-0/92 $3.00 © 1992 IEEE

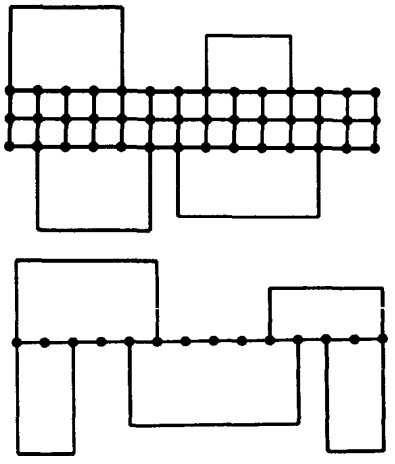

Figure 1: Channel graphs.

Definition 3 (Channel Graph Routing Problem) *A* channel graph routing problem *is a routing problem* $\mathcal{G} = (G, \mathcal{N})$ *in which*

1. *G is a channel graph, and*
2. *there are at most* $4 - \deg(v)$ *terminals of nets in* $\mathcal{N}$ *at any vertex* v *of* G.

A channel graph is embedded in the (x, y)-plane so that its vertices have integer coordinates. The x-coordinates of vertices range from x_{min} to x_{max}. In a clockwise traversal of the boundary, a corner at which you turn left is a *concave corner*. The concave corners on the boundary of a channel graph are all in the channel, and if the channel contains only one horizontal sequence of edges, then all the concave corners in the channel graph have the same y-coordinate. Otherwise, the concave corners at the top of the channel have the same y-coordinate, and similarly for those at the bottom. We classify a concave corner as one of four types, depending on its position on the boundary:

1. *top-left:* at the left side of a top sidebox
2. *top-right:* at the right side of a top sidebox
3. *bottom-left:* at the left side of a bottom sidebox
4. *bottom-right:* at the right side of a bottom sidebox

A top-left concave corner and a bottom-right concave corner are said to be of *opposite types*, and the same is true for a top-right concave corner and a bottom-left.

We use the terms *cut*, *capacity of a cut*, and *density of a cut* as they are commonly used in the literature. We write $e(X)$ for the capacity of cut X, $d(X)$ for the density of cut X, and define the *margin* of cut X, $m(X)$, as $e(X) - d(X)$. When the margin of a cut is zero, we say the cut is *saturated*, and when the margin is negative, we say the cut is *oversaturated*. Rows and columns are cuts that can be identified with straight line segments. A cut may also be identified with a polygonal line having two segments, and such a cut is referred to as a *one-bend cut*. We distinguish the two boundary edges adjacent to a boundary vertex as *pre(v)* and *post(v)*, depending on whether the edge is an entering or a leaving edge of v in a clockwise traversal of the boundary. A cut is *simple* if it contains exactly two boundary edges. The *extended degree* of a vertex is the number of edges incident with the vertex plus the number of net terminals at the vertex. An *odd vertex* is a vertex with odd extended degree. An *even routing problem* is a routing problem that contains no odd vertices. Most routing algorithms require an even routing problem as input.

Definition 4 (Neck) *Two concave corners u and v form a* neck *if they meet the following requirements:*

1. *They are of opposite type, and*
2. *there exist two simple one-bend cuts, one through* pre(u) *and* post(v), *and the other through* post(u) *and* pre(v).

Other authors would describe a neck as a simple cut through boundary edges adjacent to two rectilinear visible corners.

A channel graph routing problem P is *embedded* in channel graph routing problem P' if each net of P is also a net of P'. Our algorithm, EMBED, takes a channel graph routing problem that contains no oversaturated row, column, or neck as input, and attempts to embed the problem in an even channel graph routing problem by adding dummy net terminals to odd vertices and pairing these terminals in such a way that no cut becomes oversaturated. We prove that EMBED fails only if the original routing problem was not routable.

EMBED first partitions the channel graph routing problem into *regions* using saturated rows, columns, and necks. The algorithm then assigns dummy terminals to all odd vertices. We refer to a sequence of boundary edges between two saturated cuts (or between the two ends of one saturated cut) as a *boundary piece*.

Definition 5 (Natural Pairing) *Given a starting and a stopping point on a boundary piece, we define a* natural pairing *as a pairing performed as follows: label the dummy terminals on the boundary piece from the starting point to the stopping point* $t_1, t_2, \ldots, t_k$; *pair* t_{i-1} *with* t_i, *where* i *is even.*

Definition 6 (One-Boundary Region Set) *A* one-boundary region set *is a set of adjacent regions, each having a single boundary piece, whose boundary pieces form one contiguous piece of boundary.*

At each step, EMBED selects a saturated cut that has a one-boundary region set to one side of it, checks to see that the boundary piece of each region in the set contains an even number of dummy terminals, performs a natural pairing on each region's boundary piece, updates margins of cuts that are unowned by the region being paired, and repartitions if any cut becomes saturated. The algorithm will halt and declare the routing problem to be unsolvable if it ever finds a region that contains an odd number of dummy terminals.

The approach we take in this embedding algorithm is similar to that used by Becker and Mehlhorn[1] for planar graphs. Our selection of a one-boundary region set is equivalent to their selection of a "U-minimal cut". However, our stopping criterion allows EMBED to stop sooner than the algorithm of Becker and Mehlhorn. We stop if we discover that a region contains an odd number of dummy terminals. In order to pair all the dummy terminals in such a region, we would have to pair two dummy terminals across an already saturated cut. Thus, we stop when we discover that it is impossible to continue without oversaturating some cut. Becker and Mehlhorn stop if any cut becomes oversaturated. The complexity of the Becker and Mehlhorn algorithm is $O(bn)$. Because channel graphs are so regular in shape, we are able to obtain a complexity of $O(b)$.

3 Embedding Algorithm

Given a channel graph routing problem $\mathcal{G} = (G, \mathcal{N})$, we say that $\mathcal{G}$ is *embeddable* if an embedding of $\mathcal{G}$ in an even channel graph routing problem with no oversaturated cuts exists. As its first step, EMBED determines the margins of all rows, columns, and necks. If it discovers that some row, column, or neck is oversaturated then EMBED stops and reports that $\mathcal{G}$ is not embeddable. If no cut is oversaturated, EMBED partitions $\mathcal{G}$ into regions using saturated cuts and assigns a dummy terminal to each boundary vertex of $\mathcal{G}$ that has extended degree 3. If no cut is saturated, EMBED performs a natural pairing around the entire boundary of $\mathcal{G}$. In the remainder of this section we assume that $\mathcal{G}$ contains at least one saturated cut.

At each step, EMBED selects a saturated cut that separates a one-boundary region set from the remainder of the channel graph routing problem, and pairs the dummy terminals of each region in the set using the natural pairing technique described previously. EMBED halts if any region in the set is odd. We prove in the next section that a natural pairing of a region that has a single boundary piece and contains an even number of dummy terminals does not oversaturate any cut. Thus, the one-boundary region set can be processed by EMBED independently of the remainder of the channel graph routing problem.

Once a saturated cut has been selected and a natural pairing has been performed on its corresponding one-boundary region set, EMBED links the two boundary vertices on the other side of the cut together using a *shortcut pointer*. Any time during a natural pairing that a shortcut pointer is encountered, EMBED uses the shortcut pointer to access the next vertex. That is, during some future natural pairing EMBED will treat the shortcut pointer as a boundary edge; it will be as if the portion of the original boundary between the two vertices linked by the shortcut pointer does not exist. Figure 2 shows a channel graph routing problem after several steps of EMBED. Wherever one-boundary region sets have already been processed, two boundary vertices are linked with a shortcut pointer.

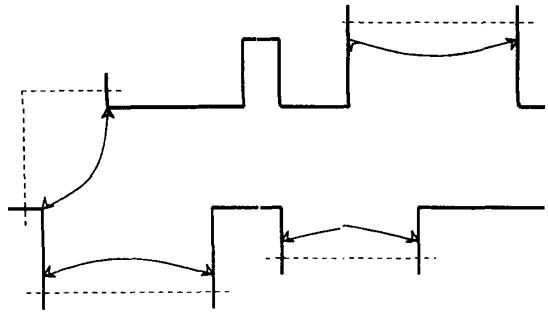

Figure 2: A channel graph after several steps of EMBED.

While performing a natural pairing on a one-boundary region set, if EMBED pairs a dummy net across an unowned cut, it reduces the margin of that cut by 1. EMBED maintains a current margin for every row, column, and neck in the graph. The current margin of a cut is its original margin reduced by the number of dummy nets paired across it so far.

A shortcut pointer may cross many cuts. If EMBED encounters a shortcut pointer during a natural pairing and the number of dummy terminals found so far is odd, then a dummy net is paired across all of the cuts crossed by the shortcut pointer. In this case, the margin of each cut goes down by one. However, it is not necessary for EMBED to reduce the margin of every cut in the group crossed by the shortcut pointer. Instead, EMBED maintains the minimum margin possessed by any of the cuts, together with a list of the cuts having this margin, and EMBED reduces this minimum margin when a dummy net crosses the group of cuts. This minimum margin and list of cuts is copied to any other shortcut pointer that crosses the same group of cuts. If the minimum margin ever goes to zero, then each cut on the list is marked saturated, as is the shortcut pointer itself. From that time on, no dummy net can cross this shortcut pointer.

EMBED selects saturated cuts in a certain order, first selecting saturated rows through sideboxes. EMBED considers top sideboxes from left to right, processing one-boundary region sets above saturated rows through these sideboxes. It then considers bottom sideboxes. Once all saturated rows through sideboxes have been processed, EMBED determines if there is a saturated

channel row, and if so, it completes the processing of the channel graph by selecting saturated channel rows, from top to bottom, and pairing the one-boundary region sets above them, until all that remains is the portion of the channel graph below the lowest saturated channel row. This portion is completed in a final pairing around its boundary.

If there are no saturated channel rows after EMBED has processed all saturated sidebox rows, EMBED selects, from among saturated columns and necks, the leftmost saturated cut, and processes the one-boundary region set to its left. In order to determine the leftmost saturated column or neck, EMBED considers the horizontal boundary edges at the bottom of the channel graph that the cuts go through. EMBED selects the cut whose lower boundary edge has, as its left vertex, the vertex with the smallest x-coordinate.

The algorithm continues selecting the leftmost column or neck at each step. Should a row through a sidebox become saturated while EMBED is processing a saturated column or neck, EMBED immediately processes the one-boundary region set outside the newly saturated row. The algorithm continues until either a saturated channel row arises, or until all that remains is the portion of the channel graph to the right of the rightmost saturated column or neck. If a channel row becomes saturated, EMBED completes the processing of the channel graph in the fashion described above. It processes saturated channel rows, from top to bottom, until all that remains is the portion of the channel graph below the lowest saturated channel row, then performs one final pairing around the boundary of this one-boundary region set. Otherwise, EMBED completes the processing with a final pairing around the one-boundary region set to the right of the rightmost saturated cut.

When EMBED has selected a one-boundary region set to be processed, it calls the following procedure, passing the cut that separates the one-boundary region set from the remainder of the graph:

```
Algorithm PROCESS_OBRS(K: cut):
begin
  for each region in the one-boundary region set do
    if the region is odd then halt
    else begin
      natural pair the region;
      update margins of unowned cuts through the
        region;
      repartition if a cut becomes saturated
    end;
  set shortcut pointer from one end of cut K to the
    other
end;
```

We now present the algorithm, EMBED, which converts the channel graph routing problem $\mathcal{G}$ to an even channel graph routing problem $\mathcal{G}'$, if it is possible to do so without oversaturating a cut:

```
Algorithm EMBED(var G: CGRP):
begin
  determine margins of rows, columns, necks;
  assign a dummy terminal to each vertex of
    extended degree 3;
  if any cut is oversaturated then halt
  else begin
    while there is a saturated row in some
    sidebox S do begin
      select the outermost saturated row R in S;
      PROCESS_OBRS(R);
    end;
    while there is a saturated row, column
    or neck do begin
      if there is a saturated row in channel C
      then begin
        while there is a saturated row in channel C
        do begin
          select the topmost saturated row R in C;
          PROCESS_OBRS(R)
        end;
        process remainder of channel graph and halt;
      end;
      else if there is a saturated column or neck
      in G then begin
        select the leftmost saturated column or neck
          K in G;
        PROCESS_OBRS(K)
      end;
    end;
    {There is only one region in G.}
    process remainder of channel graph and halt;
end;
```

4 Correctness of the Algorithm

We say that EMBED succeeds if

1. it does not halt before pairing every dummy terminal in $\mathcal{G}$, and
2. when it does halt, the channel graph routing problem contains no oversaturated cuts.

In this section we prove that EMBED succeeds in embedding $\mathcal{G}$ in an even channel graph routing problem with no oversaturated cuts, $\mathcal{G}'$, if and only if $\mathcal{G}$ is embeddable. We first develop some notation and prove some lemmas that will be useful in the proof of our main theorem of this section.

EMBED first partitions $\mathcal{G}$ over the set of saturated rows, columns, and necks, halting if $\mathcal{G}$ has any oversaturated cuts. At each step thereafter, EMBED selects a saturated cut that separates a one-boundary region set from the remainder of the problem, and attempts to process the one-boundary region set. If any region in the set is odd, EMBED fails; otherwise, EMBED performs a natural pairing on each of the regions, and updates the margins of unowned cuts. We shall use the following lemmas, whose proofs are contained in [6], to show that it is unnecessary to update the margins of cuts owned by a region.

Lemma 1 *The parity of the margin of cut X is the same as the parity of X itself.*

The parity of a cut is the parity of the number of odd vertices in the cut, or equivalently, the parity of the sum of the extended degrees of vertices in the cut.

Lemma 2 *The total number of vertices of odd extended degree in a routing problem is even.*

The *dummy load* by region $\mathcal{R}$ on cut C, $\mathrm{dl}(\mathcal{R}, \mathcal{C})$, is the number of dummy nets in region $\mathcal{R}$ that cross C. We denote the margin of cut C before a natural pairing as $m(C)$, and after the natural pairing as $m'(C)$. The following lemma proves it is unnecessary to update the margins of owned cuts after a natural pairing.

Lemma 3 *Let $\mathcal{G}$ be a channel graph routing problem with no oversaturated cuts and let there be a region $\mathcal{R}$ in $\mathcal{G}$ that contains an even number of dummy terminals and has one boundary piece. Then a natural pairing of $\mathcal{R}$'s boundary piece does not oversaturate any row, column or neck owned by $\mathcal{R}$.*

Proof. Let C be a row, column, or neck owned by $\mathcal{R}$. In a natural pairing of $\mathcal{R}$, $\mathrm{dl}(\mathcal{R}, C) \leq 2$. If $\mathrm{dl}(\mathcal{R}, C) = 1$ then, since $m(C) \geq 1$, $m'(C) \geq 0$ and C is not oversaturated. If $\mathrm{dl}(\mathcal{R}, C) = 2$ then there was an even number of odd vertices on $\mathcal{R}$'s boundary between the two ends of C making C even. By Lemma 1, $m(C)$ is an even number. Thus since $m(C) \geq 1$, we have $m(C) \geq 2$ and C is not oversaturated by the natural pairing. **QED**

Suppose $\mathcal{G}$ contains no saturated cut. Then $\mathcal{G}$ contains only one region and all cuts through $\mathcal{G}$ are owned cuts. By Lemma 2, $\mathcal{G}$ contains an even number of dummy terminals. Thus, by Lemma 3, a natural pairing around the boundary of $\mathcal{G}$ causes no cut to become oversaturated. When $\mathcal{G}$ consists of only one region, a natural pairing produces an even channel graph routing problem, $\mathcal{G}'$, with no oversaturated cuts.

During a natural pairing of a region having one piece of boundary, EMBED updates margins of unowned cuts and repartitions if any cut becomes saturated. There is no danger that some cut will become oversaturated, as the next lemma shows.

Lemma 4 *Let $\mathcal{G}$ be a channel graph routing problem with no oversaturated row, column, or neck, and let there be a region $\mathcal{R}$ in $\mathcal{G}$ that contains an even number of dummy terminals and has one boundary piece. Then a natural pairing of $\mathcal{R}$'s boundary piece does not oversaturate any unowned row, column or neck through $\mathcal{R}$.*

Proof. Let C be an unowned row, column, or neck through $\mathcal{R}$. In a natural pairing of $\mathcal{R}$, $\mathrm{dl}(\mathcal{R}, C) \leq 1$. Clearly, if dummy load is zero, C is not oversaturated by the natural pairing. If $\mathrm{dl}(\mathcal{R}, C) = 1$ then, since $m(C) \geq 1$, $m'(C) \geq 0$ and C is not oversaturated. **QED**

Each time the while loop in EMBED is executed, a cut K is selected for processing. After the one-boundary region set to one side of K is paired, the region set is effectively removed from $\mathcal{G}$ by linking the two vertices on the other side of K with a shortcut pointer. Let $\mathcal{G}_0 = \mathcal{G}$ prior to EMBED being called, and let $\mathcal{G}_{k+1}$ be derived from $\mathcal{G}_k$ by the kth iteration of the while loop in EMBED.

Here is the main theorem of this section:

Theorem 1 EMBED *succeeds in embedding a channel graph routing problem $\mathcal{G}$ in an even channel graph routing problem $\mathcal{G}'$ without oversaturated a cut, if and only if $\mathcal{G}$ is embeddable.*

Proof. ($\Rightarrow$) Assume that EMBED succeeds. Call the channel graph routing problem produced by EMBED $\mathcal{G}'$. Since EMBED succeeds, it finds no region in $\mathcal{G}$ that is odd and is able to pair all the dummy terminals in $\mathcal{G}$. Also, since EMBED succeeds, $\mathcal{G}'$ has no oversaturated cuts. This proves that $\mathcal{G}$ is embeddable.

($\Leftarrow$) We shall complete the proof by induction on the number of saturated cuts in $\mathcal{G}$. If $\mathcal{G}$ consists of only one region, then $\mathcal{G}$ contains no saturated cut. By Lemma 2, $\mathcal{G}$ contains an even number of dummy terminals. By Lemmas 3 and 4, EMBED does not oversaturate any cut in $\mathcal{G}$. Therefore, EMBED succeeds. This is the base case for our induction proof. Now assume that EMBED succeeds on any embeddable channel graph routing problem that has fewer than n saturated cuts, and consider an embeddable channel graph routing problem $\mathcal{G}$ with n saturated cuts. Since $\mathcal{G}$ is embeddable, find an embedding of $\mathcal{G}$ in an even channel graph routing problem $\mathcal{G}'$; $\mathcal{G}'$ will have no oversaturated cuts. Let K be the saturated cut first selected by EMBED for processing, and let $\mathcal{S}$ be the one-boundary region set processed in the first step of EMBED. Since K is saturated, no dummy net in $\mathcal{G}'$ crosses K. Now replace the pairing of the dummy terminals in $\mathcal{S}$ with a natural pairing. Call the resulting channel graph routing problem $\mathcal{G}''$. Consider any cut C that intersects K, i.e. any cut not owned by $\mathcal{S}$. If the portion of the boundary of $\mathcal{S}$ to one side of C is odd, a natural pairing will cause one dummy net to be paired across C. It is impossible to pair the dummy terminals of $\mathcal{S}$ without pairing a dummy net across C. If the portion of the boundary to one side of C is even, no dummy net produced by the natural pairing will cross C. Thus the margin of any cut C that intersects K is no smaller in $\mathcal{G}''$ than it is in $\mathcal{G}'$, and the number of saturated cuts through K either is not changed or is reduced.

Now remove $\mathcal{S}$ from $\mathcal{G}$. By our remark at the end of the preceding paragraph, there are fewer than n saturated cuts in $\mathcal{G} - \mathcal{S}$. We know that $\mathcal{G} - \mathcal{S}$ is embeddable since no dummy net in $\mathcal{G}'$ crosses K. Thus, by our induction hypothesis, EMBED succeeds on $\mathcal{G} - \mathcal{S}$, and consequently, EMBED also succeeds on $\mathcal{G}$. **QED**

5 Implementation of the Algorithm

The input consists of a list of (x, y)-coordinates giving the locations of the boundary vertices of the channel graph routing problem, together with the net terminals found at each vertex. The x-coordinates of the vertices range from $x_{\min}$ to $x_{\max}$. As the vertices are input they are numbered from 1 to b and inserted into a doubly-linked list that represents the boundary of the channel graph routing problem. The last vertex in the input is linked to the first to create a circular list.

Each node on the doubly-linked boundary list contains the number assigned to the corresponding boundary vertex, the (x, y)-coordinates of the vertex, the terminals at the vertex, a field that identifies the vertex as a convex corner, concave corner, or non-corner, a field that tells whether the vertex is odd, and information about the edge that immediately follows the vertex in a clockwise traversal of the boundary. We discuss the information kept on edges in the following paragraph. In order to have direct access to vertices and edges on the boundary, we keep an array of pointers to the nodes on the list indexed by the numbers assigned to the boundary vertices.

We refer to the boundary edge between vertex i and vertex $i+1$ (modulo b) as edge i; we keep information on edge i in the node for vertex i. We use the (x, y)-coordinates of vertices i and $i+1$ to determine if the edge between them is horizontal or vertical. A vertical boundary edge can be part of a row and a neck, while a horizontal edge can be part of a column and a neck. Cuts are stored in arrays; in the node representing edge i we keep the indices of cuts through edge i and a flag that tells us if any cut through edge i is saturated. With this data we can detect during a boundary traversal where one boundary piece begins and ends.

A net is represented by a pair of boundary vertex numbers. As the boundary vertices are input, a list of nets is stored in an array indexed from 1 to m. The array is large enough to accommodate any dummy nets added by EMBED.

We keep information on the columns of the channel graph routing problem in an array, *Columns*, indexed from $x_{\min}$ to $x_{\max} - 1$. Each horizontal boundary edge is either the top or the bottom of some column in the channel graph, and there is only one column for each value of x between $x_{\min}$ and $x_{\max} - 1$. For each column we keep its margin and the numbers of the edges that form the top and bottom of the column. Since the lower left corner of the channel is the first vertex that is input, the first horizontal edge encountered during data input is the top of the leftmost column. The edge that forms the bottom of the rightmost column is the first horizontal edge i such that the x-coordinate of vertex i is greater than the x-coordinate of vertex $i+1$.

We keep information on the rows of the channel graph routing problem in an array, *Rows*, indexed from 1 to r, the number of rows in the channel graph. For each row we keep the margin of the row and the numbers of the edges that form the left and right ends of the row. We use the following technique to aid in the identification of the rows of the channel graph. As we input the vertices of the boundary, we store the vertex numbers of any pairs of convex corners that are separated only by horizontal edges. A sequence of horizontal edges between two convex corners is either the top of a top sidebox, the bottom of a bottom sidebox, the top of the channel in a channel graph with no top sideboxes, or the bottom of the channel in a channel graph with no bottom sideboxes. In both of these last two cases, the x-coordinates of the convex corners are $x_{\min}$ and $x_{\max}$. We use the pairs of convex corners to find the rows of the channel graph, determining the rows in sideboxes first. Suppose convex corners numbered v and $v+k$, for $k \geq 1$ have horizontal edges $v, v+1, \ldots, v+k-1$ between them and the x-coordinates of the convex corners are not $x_{\min}$ and $x_{\max}$. Then the edges between v and $v+k$ form the top or bottom of a sidebox. We can identify the rows of the sidebox by traversing the boundary from vertex v counterclockwise and from vertex $v+k$ clockwise, until we come to a concave corner.

The rows of the channel are those rows between vertex 1 and the vertex at the upper left corner of the channel. As we input the vertices of the boundary, we save the y-coordinates of the concave corners in the channel graph. If the channel graph has no concave corners, then it is a rectangle—a channel with no sideboxes and all the rows in the graph are channel rows. If all the concave corners have the same y-coordinate, then there are no rows in the channel; all rows are in sideboxes. Otherwise, the concave corners along the top of the channel all have one y-coordinate, and those along the bottom have another. The larger of these values is the y-coordinate of the upper left corner of the channel.

We also store information on the necks in the channel graph in an array, *Necks*, indexed from 1 to s, the number of necks in the graph. A neck is defined by a pair of concave corners, one at the top of the channel and one at the bottom. For each neck we store its margin together with the number of the vertical edge adjacent to the concave corner at the top of the channel, and the number of the horizontal edge adjacent to the concave corner at the bottom of the channel in the array. In order to identify the necks in the channel graph, we create a list of all the concave corners in the graph sorted on their x-coordinates. With each concave corner we note its type (top-left, top-right, bottom-left, bottom-right). If two concave corners have the same x-coordinate and one of them is a top-left concave corner, we place it on the list first. If one of two concave corners with the same x-coordinate is a top-right, we put it on the list after the one it is tied with. Once we have a sorted list of concave corners, we traverse the list. Any time we find a bottom-left concave corner followed immediately by a top-right, or a top-left followed immediately by a bottom-right, we have identified a neck. We can create and traverse the list of concave corners in time linear in the number of columns in the channel graph. EMBED can determine all columns, rows, and necks in the channel graph routing problem in time $O(b)$.

EMBED next determines the margins of all columns, rows, and necks. To find the margin of the leftmost column, EMBED traverses the portion of the boundary to the left of the first column. EMBED uses the vertex

numbers of the terminals of each net having a terminal on this section of the boundary to determine if that net crosses the first column. The total number of nets that cross the column is the column's density. The difference between the y-coordinates of the edges at the top and bottom of the column plus one is the capacity of the column, and the margin of the column is the difference between its capacity and its density. For each column after the first, EMBED uses the density of the previous column together with the terminals located on the portions of the boundary between the two columns to calculate the margin.

A similar process is used to calculate margins of rows. For the outermost row of each sidebox, EMBED traverses the boundary outside the row. For each successive row in the sidebox, EMBED uses the density of the previous row to calculate its margin. To find the margin of the topmost channel row, EMBED makes one traversal of the boundary above this row. For successive channel rows, EMBED uses the density of the previous row to calculate its margin.

EMBED determines the margins of necks from left to right. It traverses the boundary left of the leftmost neck to determine that neck's margin. Then it uses each neck's margin to calculate the margin of the next leftmost neck, traversing the portions of the boundary between the two necks. Finding the margins of all columns, rows, and necks in the channel graph takes time $O(b)$.

If EMBED discovers that the margin of any cut is less than zero, it halts and declares the problem unsolvable. If EMBED finds a cut with a margin of zero, it marks the two boundary edges of the cut as saturated on the doubly-linked boundary list.

Finding an appropriate saturated cut inside EMBED's while loop is done using the arrays of cuts, *Rows*, *Columns* and *Necks*. EMBED first processes one-boundary region sets outside saturated sidebox rows. The array of rows contains sidebox rows before channel rows. To find the saturated rows of each sidebox, EMBED simply scans the array of rows taking note of any that have a margin of zero. Once all sidebox rows are processed, EMBED processes channel rows. Finally, EMBED processes one-boundary region sets left of saturated columns and necks. Since both the column array and the neck array are stored in left-to-right order, it is necessary only to maintain a pointer to each list to obtain the next leftmost saturated column or neck.

Once EMBED has selected a saturated cut to process, it traverses the boundary of the one-boundary region set to one side of the cut. (If the saturated cut is a row, the one-boundary region set is outside the row; if the cut is a channel row, the one-boundary region set is above the row; if the cut is a column or neck, the one-boundary region set is to the left of the column or neck.) EMBED begins its traversal at one end of the saturated cut and continues until it passes the other end of the saturated cut. As it traverses the boundary, EMBED counts the number of odd vertices it has seen. Whenever it finds an edge on the boundary that is saturated, EMBED must confirm that the count of odd vertices is even. If it is not, EMBED halts and declares the problem unsolvable. Whenever the count of odd terminals goes from even to odd, EMBED creates a new dummy net, entering information on the net into the net array. Until the count of odd terminals goes back to even, EMBED reduces the margins of all cuts through the boundary edges it traverses. During this process, if any cut's margin goes to zero, EMBED marks the boundary edges that this cut goes through as saturated. When EMBED passes the other end of the saturated cut being processed, it sets a shortcut pointer connecting the vertex at which it realized it had passes the other end of the cut with the vertex just prior to the one at which it began its traversal.

The following theorem states the main result of this section.

Theorem 2 *The embedding algorithm for channel graph routing problems,* EMBED, *has complexity* $O(b)$.

Proof. EMBED can determine all columns, rows, and necks in the channel graph routing problem in time $O(b)$. Finding the margins of these cuts also takes time $O(b)$. Finding an appropriate saturated cut inside the while loop of EMBED takes time $O(b)$, since it involves a scan of each of the arrays, *Rows*, *Columns* and *Necks*. With the use of shortcut pointers, performing natural pairings on each of the one-boundary region sets also takes time $O(b)$. **QED**

References

[1] M. Becker and K. Mehlhorn, "Algorithms for routing in planar graphs", *Acta Informatica*, Vol. 23, pp. 163–176, 1986.

[2] Michael Kaufmann and Gerhard Klär, "Routing in polygons without rectilinear visible corners", Private Communications, 1989.

[3] Michael Kaufmann and Kurt Mehlhorn, "Routing through a generalized switchbox", *Journal of Algorithms*, Vol. 7, pp. 510–531, 1986.

[4] Ten-Hwang Lai and Alan Sprague, "On the routability of a convex grid", *Journal of Algorithms*, Vol. 8, pp. 372–384, 1987.

[5] Dee Parks and Miroslaw Truszczyński, "Routing non-convex grids without holes", *Proceedings of First Great Lakes Symposium on VLSI*, pp. 157–162, 1991.

[6] Dee Parks, *Algorithms for VLSI Routing*, PhD thesis, University of Kentucky, 1991.