A Faster Algorithm for Rubber-Band Equivalent Transformation for Planar VLSI Layouts

Hsiao-Feng Steven Chen and D. T. Lee*

Department of Electrical Engineering and
Computer Science
Northwestern University
Evanston, IL 60208
e-mail:{chenstev,dtlee}@eecs.nwu.edu

Abstract

In this paper we consider the problem of transforming a single-layer topological routing of n two-terminal nets into a rubber-band equivalent using rectilinear wires in the presence of rectilinear circuit modules. Given a topological planar VLSI layout sketch with $|F|$ features and $|W|$ non-crossing wire segments connecting n two-terminal nets, we present an $O(|F| \cdot |W|)$ time algorithm to do the vertex-disjoint rubber-band equivalent transformation of these n nets if it exists. The algorithm consists of two phases, computing a loose homotopy with four spokes matrices, and computing a vertex-disjoint rubber-band equivalent of the given homotopy, each phase taking $O(|F| \cdot |W|)$ time and space. Both complexities are asymptotically optimal in the worst case. From the vertex-disjoint rubber-band equivalent of the given homotopy, one can obtain the detailed routing within the same time complexity. Experimental results are also presented.

*Supported in part by the National Science Foundation under the Grant CCR-9309743, and by the Office of Naval Research under the Grant No. N00014-93-1-0272.

1 Introduction

Planar or single layer routing problem is to wire a set of nets with a given topology using a certain routing model such that the wires belonging to different nets form vertex-disjoint paths. This problem has been shown to be NP-complete[16], and is unlikely to have a polynomial time solution. Only planar routing of restricted classes were known to be solved efficiently[1] [11] [5] [10].

Among them is a problem, known as *Detailed Routing given a Homotopy* (*DRH*). This was first defined by Pinter[14]: given a set of rectangular modules within a bounding box, numbered terminals on the modules' boundaries, and a homotopy (rough routing) specification of each net, is there a one-layer detailed routing for this configuration that conforms to the given homotopy? Such a homotopy can be obtained by, for example, the techniques given in [3] and [17]. As studied in [1][5] and generalized to a so-called *sketch* denotation in [10], the *DRH* problem has been shown to be solvable in polynomial time.

Traditionally, in previous results [10], a *DRH* problem is solved by firstly testing the legality of the given sketch (the circuit layout) in terms of *capacities* and *wire densities*(number of passing wires), secondly computing *rubber-band equivalent*(*RBE*) and finally transforming it into a detailed routing. The correctness of the detailed routing algorithm is assured by the routability testing in the beginning. The theoretical basis for this routability testing was first established by Cole and Siegel[1] and later improved by Leiserson and Maley [10](Lemma 2.1 in this paper.) Leiserson and Maley proposed polynomial algorithms for both routability testing and detailed routing by applying as preprocessing *RBE* transformation and computing the *spokes* for the *RBE*. However, as pointed out in [4] computation of spokes is not as straightforward as described in the original paper[10]; more than one pass over the features may be needed.

To remedy the problem Dai, et al.[4] proposed a rubber-band sketch, called *extended rubber-band equivalent* (which will be referred to as *ERBE* in the following) to create spokes explicitly, updating the given sketch dynamically as they are created. They then perform routability test using geometric algorithms such as convex hull, range search and Delaunay triangulation[15]. It is not clear though how the *convex hull* of the features can be used to help compute the spokes as each net may have a very complex wiring topology. No complexity analysis was given of their proposed algorithm. It appears that dynamic updating of *RBE* can be costly computationally. Our approach to the *RBE* transformation is similar to *ERBE*, except we in a single phase compute the spokes in two passes over the given sketch, and perform the *RBE* transformation in another; routability test is done simultaneously.

Planar routers given a homotopy have also been applied to many other important layout performance optimization phases such as compaction with jog insertion [13] [12] and performance driven routing for multichip module (MCM) layout [3] [2]. In particular, finding the *RBE* of the given homotopy is a backbone not only for the detailed planar routing, but also for many *VLSI* layout applications. It is thus most desirable to have a more efficient algorithm to perform this task.

In this paper, we propose a more efficient and easy-to-implement algorithm for *RBE* transformation. The main result is the computation of *loose homotopy* which records the given topological information more efficiently. We then use that information to obtain a *vertex-disjoint RBE* (*VRBE*) of the homotopy, and construct the detailed routing given the *VRBE*. The rest of the paper is organized as follows. We first provide some definitions and common terminology in section 2, which will be used in later discussions. In section 3, we describe our *VRBE* transformation algorithm along with algorithms for routability test, and solving *DRH*. Finally, we give in section 4 an extension to multi-terminal nets and provide some experimental results from our implementation.

2 Preliminaries and Previous Work

2.1 Components of a Sketch

The input of a topological layout router is usually represented by a *sketch*[10] [13]. As is shown in Fig. 1(a), a sketch $S = (F, M, W)$ is a planar drawing of a physical layout in the grid, where F is a finite set of *features*, which are points or open line segments on the boundaries of the modules m_i's in M, and W is a finite set of *wires*, which are vertex-disjoint simple paths connecting the point features by a sequence of straight line segments. In Fig. 1, the dark lines and circles stand for the features, and dotted lines for the boundaries of the modules.

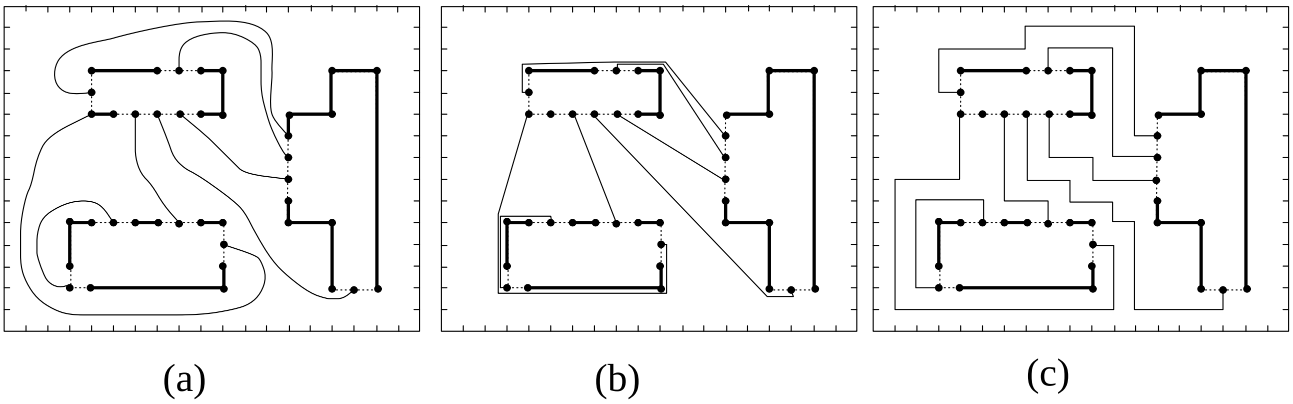


Figure 1: (a) Layout specification, (b) rubber-band equivalent and (c) detailed routing

To simplify the discussion, wires are assumed to have exactly two endpoints (point features) which are called *terminals*. We assume there are n two-terminal nets in the sketch, and all the features lie on a grid.

2.2 Homotopy of Wires and Detailed Routing

We are interested in finding a detailed routing of these n nets such that the wire segments of each net must lie on the grid without overlapping and the routing has a minimum length among those that are homotopic to the given sketch. More formally a detailed routing of a sketch $S = (F, M, W)$, $W = \{\mu^1, ..., \mu^n\}$, is a sketch $S^* = (F, M, W^*)$, $W^* = \{\nu^1, ..., \nu^n\}$, such that ν^i is *homotopic* to μ^i, i.e., net μ^i and net ν^i have the same pair of terminals, and μ^i can be transformed continuously into ν^i without moving its endpoints and without allowing its interior to touch a feature in F, and such that the ν^i's are vertex-disjoint paths in the rectangular grid of unit-spacing, where n represents the total number of nets. Moreover, W^* should be a routing that optimizes both individual and total wire length. The detailed routing of Fig. 1(a) is shown in Fig. 1(c). Fig. 1(b) shows a traditional *RBE* which is generally not vertex-disjoint.

To specify the homotopy of a net, μ^j, in W, we denote it as $\mu^j =< s_0^j, s_1^j, ..., s_l^j >$ for some l, where (s_k^j, s_{k+1}^j) denotes wire segment w_k^j and $|W| = \sum_j \sum_k w_k^j$. That is, the size of the wire homotopy is the total number of wire segments and these wire segments are the basic units from which we extract the homotopy information to obtain a vertex-disjoint *RBE*, as we describe in the following section.

2.3 Routability Condition

Before the routing procedure is invoked, it is necessary to check if the wires going through the channels between the modules are properly assigned by the topology. That is, to find if there is any channel with insufficient space for the wires to pass through. We shall follow the terminology given in [10]. A sketch is *legal* if (1) every feature lies on the grid, and every terminal lies on a grid point and (2) the wires in a sketch are vertex-disjoint paths on the grid. Let $c = (x_c, y_c)$ and $d = (x_d, y_d)$ be two points in F, and let $\overline{cd}$ denote the open line segment between c and d. A *cut* $\overline{cd}$ is an open line segment, which intersects no features in F. In other words, points c and d are *visible* to each other. A cut $\overline{cd}$ is *critical* if either (1) both c and d are point features, or (2) one of them is the endpoint of a feature, and the other is the point in F closest to c. The *capacity* of a cut is defined as the *maximum* number of legal wires that can cross the cut. More precisely, for a cut $\overline{cd}$, its capacity is given as $cap(\overline{cd}) = \max\{|x_c - x_d|, |y_c - y_d|\} - 1$, or $\max\{\Delta x, \Delta y\} - 1$. The actual number of wires of the sketch crossing a cut is called the *density* of the cut. A cut is said to be *safe*, if its density is not greater than its capacity, and *saturated* if they are equal. The following lemma is proved in [1] and [10].

Lemma 2.1 *A sketch $S = (F, M, W)$ is routable if and only if $(F, M, \emptyset)$ is legal and every critical cut in $S = (F, M, W)$ is safe.*

A *diagonal cut* $\overline{cd}$ is a cut emitted from point or corner feature c of some module in M to d with slope $+1$ or -1, where d is a point feature or a point on a line feature such that it is the first point hit by the cut emitted out of c. Associated with each point feature f_i there are at most four diagonal cuts. In the following, we use $f_i(\theta)$, $\theta = k\pi/4, k = 1,3,5,7$, to denote the four diagonal cuts with respect to feature f_i in θ direction. We will discuss this sort of cuts in detail later, since they play an important role in routing.

Using Lemma 2.1, the authors of [1] and [10] proposed a two-stage method in which routability test is done in the first phase, and a detailed routing is computed in the second phase. Specifically it is shown in [10] that transformation of the sketch into its *RBE* form takes $O(|F| \cdot |W| \log |F|)$ time and $O(|F| \cdot |W|)$ space. Routability test can then be performed on the *RBE* representation in $O(|F|^2 \log |F|)$ time. The second stage, which is also based on the *RBE* form can be accomplished in $O(|F| \cdot |W| \log(|F| \cdot |W|))$ time. We show however that all the stages, computing *RBE*, testing and routing, can be done simultaneously, as stated in the following theorem, which is the main result of this paper.

Theorem 1 *Computing RBE, testing the routability and routing of a planar VLSI layout can be done in time $O(|F| \cdot |W|)$, which is asympototically optimal in the worst case.*

3 The Vertex-Disjoint RBE Transformation Algorithm

As mentioned earlier, the objective of the detailed routing is to find a (vertex-disjoint) routing on the grid or a *legal* sketch, with minimum wire length. It is obvious that "tightening" of the wires in the *RBE* framework gives a lower bound on the total length of the routing. Note however that in previous approaches the computed *RBE* paths for the nets are no longer vertex-disjoint and that this transformation itself already takes $O(|F| \cdot |W| \log |F|)$ time and $O(|F| \cdot |W|)$ space. A key step here is to maintain the order of wires that touch and go around a corner feature. To obtain a more efficient algorithm asymptotically we use a different approach. Before we describe our approach let us make a few observations. In the routing stage, the precise distance of wires from the corner features, against which a number of paths are tightened resulting from the *RBE* transformation, must be computed. More precisely let f_i denote a corner feature and w the kth wire touching against f_i. Then w must cross a diagonal cut $f_i(\theta)$ (or a line containing $f_i(\theta)$) at a point q whose distance from f_i is $|\overline{f_i q}| \geq \sqrt{2}k$. The segment $\overline{f_i q}$ with respect to wire segment w is called an *obstructing spoke* for w, i.e., the final wiring homotopic to w must not *cross* the interior of this spoke. If the final wiring homotopic to w passes through q such that $|\overline{f_i q}| = \sqrt{2}k$, the spoke is referred to as a *supporting spoke* and q is a *supporting vertex* of w. A spoke is positive if its slope is $+1$, and negative if -1. In other words the detailed routing for each wire $w \in W$ is found by pushing the rubber-band equivalent of each wire segment away from the features to avoid its positive or negative obstructing spokes. It follows that if the length of the obstructing spoke for any wire is *greater* than the actual length of the diagonal cut, the layout is *not* routable, for the density of the cut exceeds its capacity. Since each wire segment can have $O(|F|)$ obstructing spokes, to find the positions of all spokes takes at least $\Omega(|F| \cdot |W|)$ time[10] in the worst case. We'll show below how the obstructing spokes for all the wires can be computed in $O(|F| \cdot |W|)$ time and $O(|F| \cdot |W|)$ space, which is thus asympototically optimal in the worst case.

We adopt a similar transformation, called *vertex-disjoint RBE*. From the *vertex-disjoint RBE* we may compute the traditional *RBE* while preserves the wire ordering against certain feature, perform routability testing or obtain an optimal detailed routing in $O(|F| \cdot |W|)$ time. Since the ordering of *RBE* is kept, the performance optimization phase using dynamic updating[2] can then be applied as well. To summarize, our vertex-disjoint *RBE* transformation algorithm has the following phases:

Algorithm VERTEX_DISJOINT_RBE

Input: A sketch $S = (F, M, W)$

Output: The vertex-disjoint RBE corresponding to S.

Method:

1. Extract the loose homotopy W' from W:

   The result is a representation, called *spokes matrix*, of spokes for each wire segment with respect to each point or corner feature.

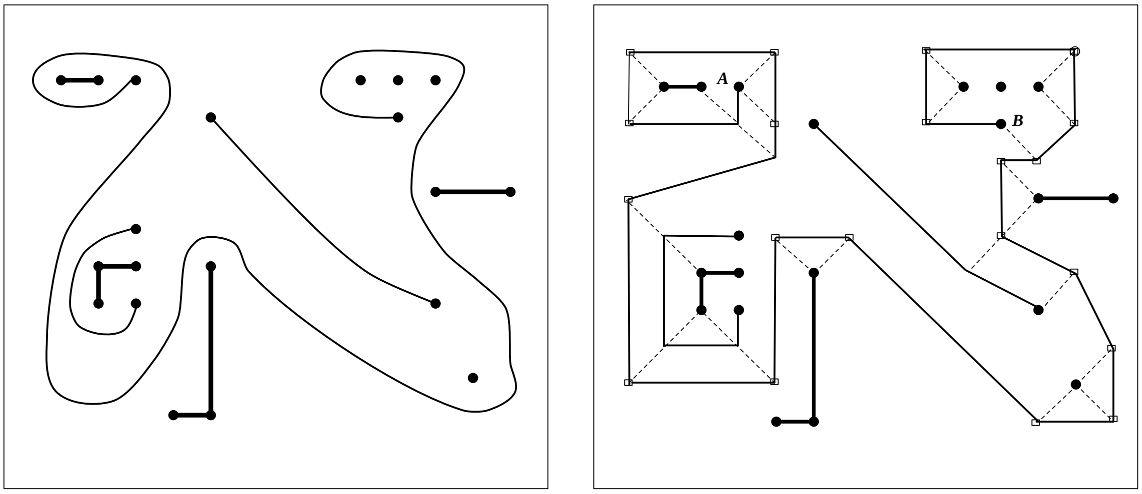
2. Compute vertex-disjoint *RBE* W'' from W':

   Note that according to the definition of the vertex-disjoint *RBE*, the new wire homotopy consists of sets of chains of segments of 0, ∞, positive or negative slopes.

A problem instance and its vertex-disjoint RBE are shown in Fig. 2. Each of the steps will be explained in detail in the subsequent subsections.

3.1 The Spokes Matrix

In this section, we define a basic data structure, *spokes matrix*, denoted $Matrix_\theta(i, j_k)$. Entry $Matrix_\theta(i, j_k)$ represents the exact distance in units of grid of the *real* intersection between diagonal cut $f_i(\theta)$ and wire segment w_k^j of net μ^j, for some k. Such distance is determined according to the order of the intersections between the diagonal cut $f_i(\theta)$ and wire segment w_k^j along $f_i(\theta)$. The



(a) (b)

Figure 2: (a) A topological layout instance and (b) its vertex-disjoint RBE

distance is defined as one plus the number of nets that intersect $f_i(\theta)$ *non-redundantly* between f_i and the current intersection. If w_k^j is the qth wire passing through diagonal cut $f_i(\theta)$, then $Matrix_\theta(i, j_k) = q$. Entry $Matrix_\theta(i, j_k)$ gives the *lower bound* of the distance between μ^j and feature f_i in the direction θ in the final routing.

We need four matrices for the four possible orientations of the diagonal cuts: $Matrix_\theta(i, j_k)$, where $\theta = l\pi/4, l = 1, 3, 5, 7$. The size of the matrix[1] is $O(|F| \cdot |W|)$, since $\sum_j \sum_k w_k^j = |W|$. These matrices can be obtained efficiently by scanning in four directions over all the features. We'll describe its computation later.

First we note that when a net passes through a diagonal cut more than once, it may form some empty "*loop*" with the cut, creating *redundant* intersections. These intersections are not *real* in the sense that they can be made non-existent with a homotopic transformation, *e.g.,* tightening the wires at consecutive crossings will remove the "loop". For example in Fig. 3, the two intersections a_1 and a_2 are redundant, as the loop can be eliminated by a "*tightening*" operation. Of course if there were features inside, the loop cannot be eliminated. The procedure described below distinguishes a non-redundant loop from a redundant one.

3.2 Finding Real Intersections

Consider a cut $f_x(\theta)$, induced by some feature f_x along θ direction ($\theta = \pi/4, 3\pi/4, 5\pi/4$ or $7\pi/4$), positioned horizontally and let this cut be l for simplicity. Suppose net μ^j intersects l more than once. Let the intersections be ordered from left to right as $a_1, a_2, \ldots, a_k$. The subnet between two

[1]Technically each entry $Matrix_\theta(i, j)$ is a list of nodes, in which the kth node corresponds to wire segment w_k^j. We use $Matrix_\theta(i, j_k)$ to denote such a node.

consecutive intersections, a_i and a_{i+1}, $i = 1, 2, \ldots, k-1$, along with segment $\overline{a_i, a_{i+1}}$ is referred to as a *pocket*, denoted $pocket_i^j$, and segment $\overline{a_i, a_{i+1}}$ is referred to as the *opening* of $pocket_i^j$. $pocket_i^j$ is called an *up* pocket if it lies *above* the cut and a *down* pocket otherwise. These pockets, $pocket_i^j$, $i = 1, 2, \ldots, k-1$, are alternately up and down pockets. (Fig. 3) A pocket is *redundant* if it does not contain any point feature and *non-redundant* otherwise. Note that a redundant pocket $pocket_i^j$ of net μ^j may contain pockets, $pocket_{i'}^{j'}$ of other nets $\mu^{j'}$, which are necessarily redundant. In this case the opening of $pocket_i^j$ totally contains the opening of $pocket_{i'}^{j'}$, i.e., segment $\overline{a_{i'}, a_{i'+1}}$ is a subset of segment $\overline{a_i, a_{i+1}}$. This is called *nesting* property. The endpoints, a_i and a_{i+1}, of the opening of a redundant pocket $pocket_i^j$ are said to be *fictitious*, and the remaining intersections not classified fictitious are said to be *real*. If net μ^j intersects cut l only once, the intersection is *real*. The goal is to find for all nets all the *real* intersections on each diagonal cut.

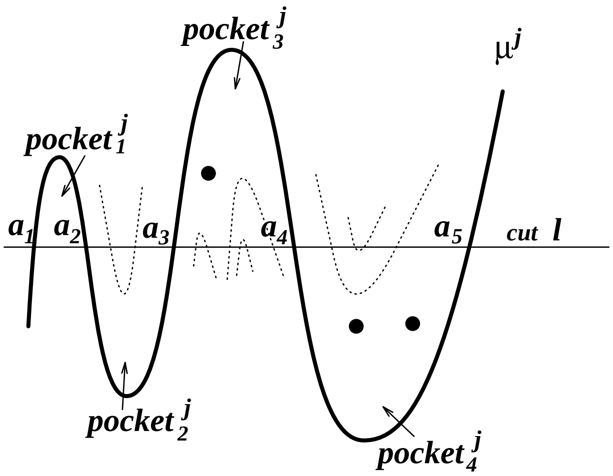


Figure 3: Illustration of pockets and the nesting property

Suppose we label the intersections of diagonal cut l and each net by the name of the net and order them from one endpoint of the cut to the other. If cut l is emanated from a terminal of net μ^j, then the labels of the intersections will begin with label j. We shall scan the labels from one endpoint of the cut to the other. Assume that for each pocket if it contains a point feature, the point feature must be a terminal of a net and the net must intersect the cut. In this case it is easy to see that if two identical labels t occur adjacently, then net μ^t has a redundant pocket with opening being the segment defined by these two intersection points, and both labels can be removed, as they are fictitious intersections. Due to the nesting property, two labels t' that do not appear adjacently on the cut may become adjacent as a result of the removal of fictitious intersections. This corresponds to the case where a redundant pocket is contained inside another (redundant) one. Then these two labels can also be removed. From the above discussion we describe the algorithm below.

Algorithm INTERSECT(l)

Input: Intersections of nets with cut l ordered from one endpoint of l to the other and maintained as a doubly-linked list L, with pointers *next*(v) and *prev*(v) to the next and previous elements of node v respectively. Assume that *ListHead* contains a *dummy* label, say 0, and the next element of the last node in L is *nil*.

Output: A new list containing elements that correspond to *real* intersections.

Method:

1.$A \leftarrow$*ListHead*, $B \leftarrow$*next*(A).

2.**while** $B \neq$ *nil* **do**

if $A \neq B$

then (Comment: Advance.)

$A \leftarrow B$, $B \leftarrow$ *next*(A).

else (Comment: Back up)

$A \leftarrow$ *prev*(A),

delete *next*(A) and B,

$B \leftarrow$ *next*(A).

3.**end**

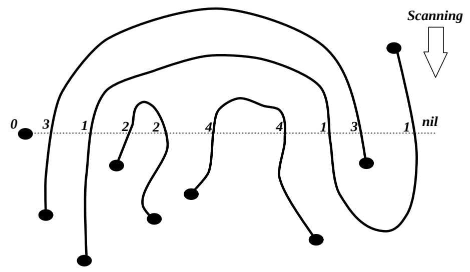


Figure 4: Removing fictitious intersections

Let us refer to (Fig. 4) for an illustration of this algorithm. Suppose the list $L = \{0, 3, 1, 2, 2, 4, 4, 1, 3, 1,$ *nil* $\}$. First deletion occurs when $A = 2, B = 2$. The new list $L_1 = \{0, 3, 1, 4, 4, 1, 3, 1,$ *nil* $\}$ and $A = 1, B = 4$ after backing up. We then advance to $A = 4, B = 4$. After deleting A and B, we obtain $L_2 = \{0, 3, 1, 1, 3, 1,$ *nil* $\}$ and $A = 1, B = 1$. Continue deleting *next*(A) and B we have $L_3 = \{0, 3, 3, 1,$ *nil* $\}$ and $A = 3, B = 3$. Finally we have $L_4 = \{0, 1,$ *nil* $\}$ and $A = 1, B =$*nil*. Only one real intersection remains in the list.

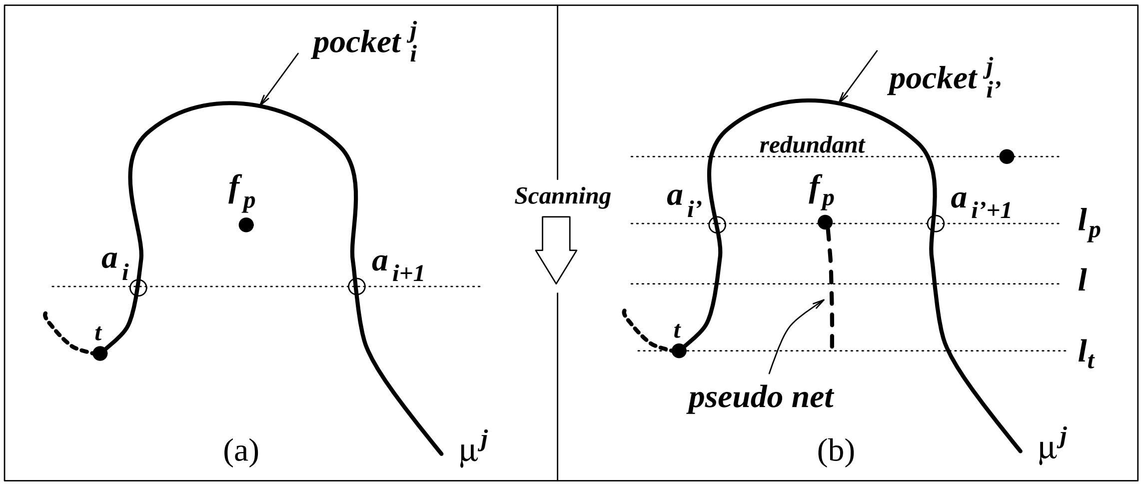


Figure 5: Introducing a pseudo net.

Obviously **Algorithm** INTERSECT takes time proportional to the size of the list L.

We now consider the problem of constructing the doubly-linked list L_l for each cut l. At first glance we may be tempted to simply compute the intersections of l with all nets and order these intersections on l into list L_l. Unfortunately this simple-minded approach does not work, since it does not distinguish between redundant and non-redundant pockets. See for example Fig. 5(a). When an isolated point feature f_p exists in pocket $pocket^j_i$, calculating simply the intersections of l and all other nets does not detect the presence of f_p. To cope with the problem, we shall introduce a *pseudo* net (Fig. 5(b)) by *inserting* an extraneous label in the opening of each non-redundant pocket for net μ^j so that the two labels that correspond to the two endpoints of the opening do not appear adjacently and hence will not be removed by **Algorithm** INTERSECT(l). Now let us describe how we determine when a pseudo net needs to be introduced. Refer to Fig. 5, where $pocket^j_i$ contains an isolated point f_p. Let $l_p = f_p(\theta)$ be the sweepline parallel to l and passing through f_p. Net μ^j has an up pocket $pocket^j_{i'}$ with opening $\overline{a_{i'}, a_{i'+1}}$, which contains f_p. Thus for l_p $pocket^j_{i'}$ is non-redundant. In fact for sweeplines above l_p the corresponding pocket is redundant, and for sweeplines between l_p and $l_t(= f_t(\theta))$ the corresponding pocket is non-redundant, where t is a terminal of net μ^j. In other words, we may introduce a pseudo net from sweepline l_p to sweepline l_t. Below sweepline l_t, terminal t will play the role of point f_p, and a pseudo net may need to be introduced below l_t.

Now we describe a top-down plane sweep algorithm[15]. The bottom-up sweep is symmetrical. Here we assume that the *horizontal* sweepline corresponds to a line of slope +1. Namely, the top-down scan corresponds to scanning points (x_i, y_i) in descending order of $y_i - x_i$. We first process each net and identify all the extremes, local maxima and minima. These extremes and all point features will be our *event* points. For the *status* of each sweepline we maintain a height balanced binary tree T recording all the intersection of the nets with the current sweepline. A terminal $t = (x_t, y_t)$ of each net is considered a local maximum or minimum depending on whether the other endpoint of the wire segment incident on t is below or above the sweepline passing through t, respectively.

The operations on the status of sweepline associated with each event point are quite standard, i.e., insertions and deletions of wire segments associated with various nets into and from the height balanced tree should be performed to maintain the order of intersections with the current sweepline. We describe other actions that need to be performed specific to this problem. When a maximal terminal t of net μ^j is encountered we copy a subset of T into a doubly-linked list L_{l_t} of nodes whose labels correspond to the intersections of the nets with the cut l_t, which is a subsegment of the current sweepline. A node j is inserted into T between nodes j_l and j_r whose corresponding wire segments lie to the left and to the right of wire segment of net μ^j incident on t. When a minimal terminal b of net μ^j is encountered, the current contents of T on the cut $l_b(= f_b(\theta))$ is recorded in list L_{l_b}. The node j in T corresponding to the wire segment terminating at b will be replaced with a pseudo node. If its left or right neighboring node is a pseudo node, we delete the neighboring pseudo node. This will ensure that no two pseudo nodes will occur consecutively in T. Each time when a point feature f_p is encountered for which a cut $l_p(= f_p(\theta))$ is of interest, we'll copy the contents of T associated with the cut l_p in list L_{l_p}. And a pseudo node is generated if f_p is a local minimum.

After the bottom-up sweep is performed we have for each diagonal cut l a doubly-linked list L_l which contains the intersections of all nets with cut l plus pseudo nodes, if any, inserted at appropriate places. We then invoke **Algorithm** INTERSECT(l) to obtain a list $List_l$ of *real* intersections.

Note that the number of added pseudo nets does not exceed $O(|F|)$ due to planarity property. The above scanning procedure takes $O((|F| + |W|)\log(|F| + |W|))$ time, excluding time for copying subsets of T into doubly-linked lists L_l associated with each cut l. The total amount of time and space involved in computing doubly-linked lists associated with all diagonal cuts is $O(|F| \cdot |W|)$. Once the doubly-linked list L_{l_i} associated with feature f_i, say in direction $\theta = \pi/4$ is available, we can compute the entries of the spoke matrix $Matrix_\theta(i, j_k)$ as follows. Scanning $List_{l_i}$ from head to tail, if wire segment w_k of net μ^j is the qth real intersection with cut l_i, i.e., the qth entry in $List_{l_i}$, $Matrix_{\pi/4}(i, j_k) = qD$, where D denotes the minimum spacing between wires. ($D = 1$ is assumed in the grid model.) $Matrix_{\theta'}(i, j_k)$ for other orientation θ' can be computed similarly. Obviously, this computing time is proportional to the total number of real intersections, i.e., $O(|F| \cdot |W|)$. We remark that when calculating the *distance* between the real intersection of wire segment w_k^j and diagonal cut $f_i(\theta)$, where $\theta = \pi/4, 3\pi/4, 5\pi/4$ and $7\pi/4$ we also record such an intersection in a node n_i^j in the list $\mathcal{L}(\mu^j)$ associated with net μ^j; such a node is shown as a small square along the net in Fig 7. Notice that the order of these intersections is well-defined along the net so we later can compute the shorest path easily by traversing the list $\mathcal{L}(\mu^j)$. As a matter of fact, this list $\mathcal{L}(\mu^j)$ of intersections is homotopic to the original homotopy of the given net μ^j.

3.3 Extracting the Sleeve for the Given Homotopy

In a triangulated simple polygon P, the associated *dual graph* of the triangulation $T = (V, E)$ such that each node of V corresponds to a triangle of the triangulation and each edge of E connects two nodes of V if and only if the corresponding two triangles share a diagonal of P is a *tree*. The diagonal of P and the corresponding edge in T are said to be *dual*. As is also defined in [9], a *sleeve* is a triangulated polygon whose dual graph is a *path*. It has been shown[9] that a shortest path between two vertices s and t that totally lies in the interior of P must be restricted to a sleeve, whose boundary is composed of a left chain and right chain connecting s and t. Such a sleeve P' is shown in Fig. 6.

It turns out that the associated vertex-disjoint *RBE* W'' can be obtained by finding for each net μ^j the shortest path in its corresponding sleeve defined by the induced spokes of the diagonal cuts intersected *non-redundantly* by net μ^j. Assume the homotopy of a net from terminal s to t is a directed path. The vertices of its associated sleeve are determined by the vertices of the obstructing spokes from either the left side or the right side of the path; the precise positions are determined by the spokes matrices, which provide the information about the supporting vertices with respect to the position of the features. A sleeve formed by negative spokes for a net and the shortest path between s and t are shown in Fig. 7.

In the following, we explain how the vertices of the sleeve associated with net μ^j can be computed from the list $\mathcal{L}(\mu^j)$. Recall that each node in the list contains the information about which feature f_i presents an obstructing spoke for μ^j and the length of the obstructing spoke is recorded in $Matrix_\theta(i, j)$. Refer to Fig. 7 which shows spokes in directions $3\pi/4$ (to the left of μ^j) and $7\pi/4$ (to the right of μ^j). For each node in $\mathcal{L}(\mu^j)$ we compute the boundary points consecutively as follows. If the node (the small empty squares shown in Fig. 7) was introduced by a feature f_i from "left", the coordinates, $(x(P), y(P))$, of the vertex on the left boundary are determined by $x(P) = x(f_i) -\ Matrix_{3\pi/4}(i, j)$ and $y(P) = y(f_i) +\ Matrix_{3\pi/4}(i, j)$. If it is from "right", $x(P) = x(f_i) +\ Matrix_{7\pi/4}(i, j)$ and $y(P) = y(f_i) -\ Matrix_{7\pi/4}(i, j)$. The sequence of vertices on the left (respectively, right) forms the left (respectively, right) boundary of the sleeve. Note these left and right chains of vertices may coincide resulting in a single chain with the sleeve having empty interior. It is important to note that the order in which the boundary vertices occur is the same as the order on the nodes that appear in the list $\mathcal{L}(\mu^j)$. To sum up, our approach to computing W'' is to first construct the sleeves by traversing the list $\mathcal{L}(\mu^j)$ for each net μ^j, then apply the available algorithm in [9] to find the shortest path in the sleeve. In our implementation we do not explicitly construct the sleeve. Instead, we apply the *funnel* algorithm [9] to compute the shortest path directly as we traverse the list $\mathcal{L}(\mu^j)$ for each net μ^j. W'' can be decomposed into two components: W''_+ and W''_-; spokes matrices in directions $\pi/4$ and $5\pi/4$ determine W''_+ (avoiding positive obstructing spokes), and spokes matrices in directions $3\pi/4$ and $7\pi/4$ determine W''_-. Our algorithm first computes W''_+ and W''_- and then merges them to obtain the vertex-disjoint *RBE*. The computation of W''_+ and W''_- is similar to the routing algorithm (*Phase 1*) described in [10], and is omitted here (in [10], they defined a term *corridor* similar to the sleeve denotation here but with the features instead of the obstructing spokes as the boundary). The interested reader is referred to the original article[10]. We remark here that the time complexity for this shortest path computing is proportional to the number of the vertices of the sleeves. In other words, it can be done in $O(|F|\cdot|W|)$ time by traversing all the features since there are at most that many obstructing spokes for the sketch.

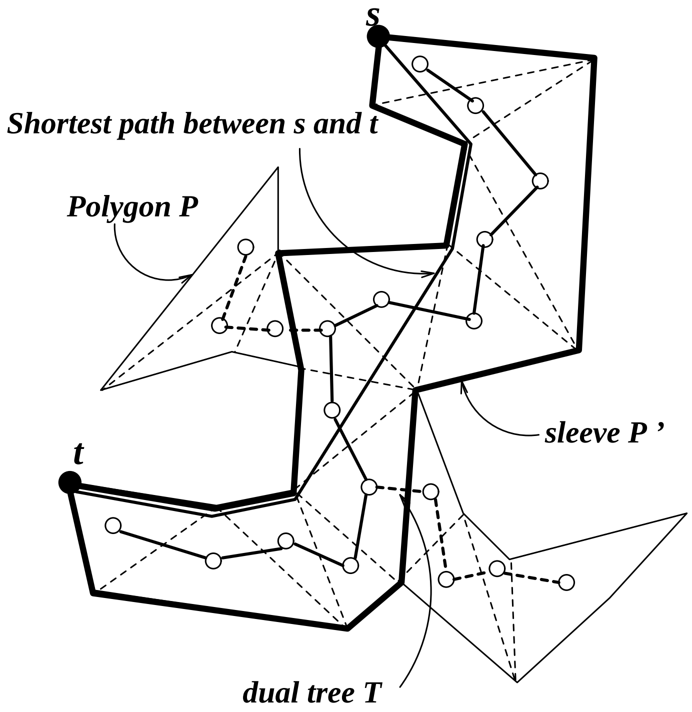


Figure 6: A sleeve in a triangulated simple polygon

For the problem instance in Fig. 2(a), the sleeve for net connecting A and B is shown in Fig. 8(a), and its W''_+ and W''_- are shown in (b) and (c), respectively. The shortest path for W''_+ is also shown in Fig. 8(b) and the merged W'' for the net connecting A and B is shown in Fig. 2(b).

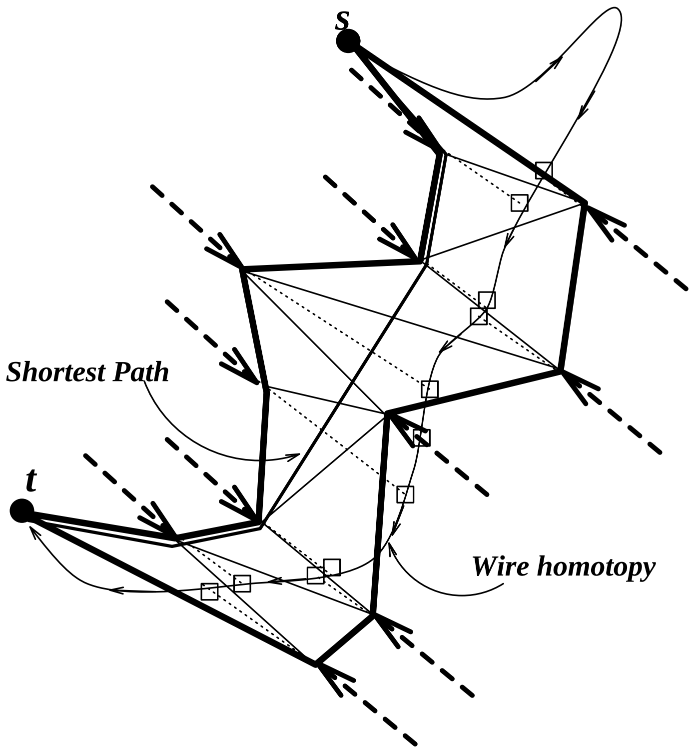


Figure 7: Boundary of a sleeve formed by obstructing spokes.

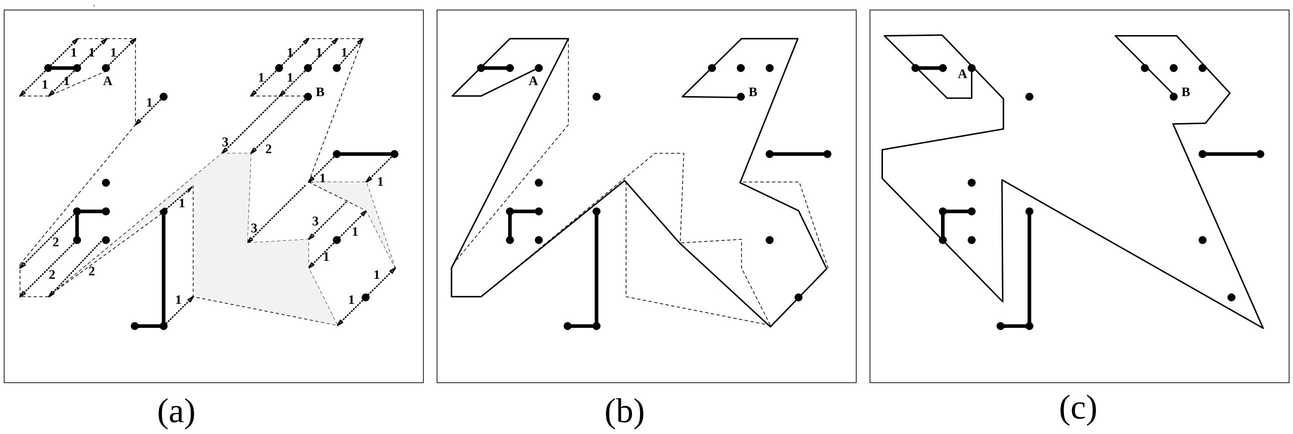


Figure 8: Transformation to (b) W''_+ and (c) W''_-

3.4 Merging W''_+ and W''_-

Let μ^j_+ denote the net in W''_+ that is homotopic to net μ^j, and μ^j_- is defined similarly. To merge W''_+ and W''_-, an observation has been made in [10] that a supporting vertex z of any net μ^j_+ in W''_+ or net μ^j_- in W''_- is *good* if the corresponding supporting spoke makes angles of less than 135° with the wire segments incident on z. Otherwise, it is *bad*. The resulting net $\mathcal{N}^j$ in W'' homotopic to net μ^j has the same set of good vertices in both μ^j_+ and μ^j_-. The set of *good* vertices of μ^j_+ and μ^j_- can be identified by traversing net μ^j following the above characterization rule. The resulting $\mathcal{N}^j$ in W'' is simply the sequence of *all* good vertices thus identified on nets μ^j_+ and μ^j_-. Note that since $\mathcal{N}^j$ is *homotopic* to net μ^j, all the good vertices of both μ^j_+ and μ^j_- occur in the same order as the real intersections do on net μ^j that were found in Step 1 of **Algorithm** VERTEX-DISJOINT_RBE and stored as nodes n^j_i in the list $\mathcal{L}(\mu^j)$. The collection of nets $\mathcal{N}^j$ for each μ^j forms the set W''.

After this phase, each net consists of a series of routed part (i.e., wire segments with either 0 or ∞ slope) interleaved with monotone chains (i.e., with either positive or negative slope other than 0 and ∞). In other words, net $\mathcal{N}^j$ is represented by $\mathcal{N}^j =< u^j_0, u^j_1, ..., u^j_{k'} >$, where u^j_0, $u^j_{k'}$ are the terminals of net $\mathcal{N}^j$, and each point u^j_l has the following two attributes:

- $feature(u^j_l)$: the feature that forms the supporting point of the monotone chain.
- $posi(u^j_l)$: the x- and y-coordinates of u^j_l determined by the spokes matrices.

Based on the result [10] we know that, if the sketch is routable, by using the above algorithm we can obtain a vertex-disjoint *RBE*. We summarize in the following lemma.

Lemma 3.1 *Given a routable sketch, its vertex-disjoint RBE can be computed in* $O(|F| \cdot |W|)$ *time.*

3.5 Routability Testing

We shall show that after W'' is computed for each net, a vertex-disjoint *RBE* form exists if and only if the given sketch is routable. It suffices to show that if the sketch is not routable, then the resulting W'' is not vertex-disjoint. Recall that in Lemma 2.1 sketch $S = (F, M, W)$ is routable if and only if $(F, M, \emptyset)$ is legal and every *critical* cut in (F, M, W) is safe. Note that if any of the diagonal cut violates the safety condition, i.e., the density exceeds its capacity, the sketch is not routable. This condition can be detected in the first phase of the **Algorithm** VERTEX-DISJOINT_RBE, when the length of any obstructing spoke exceeds that of the diagonal cut. We thus assume from here on that this trivial case does not arise. For each net $\mathcal{N}^j$ in W'' we say that a vertex v is *illegal* if one of the wire segments incident on v makes an angle of less than $\pi/4$ with its supporting spoke.

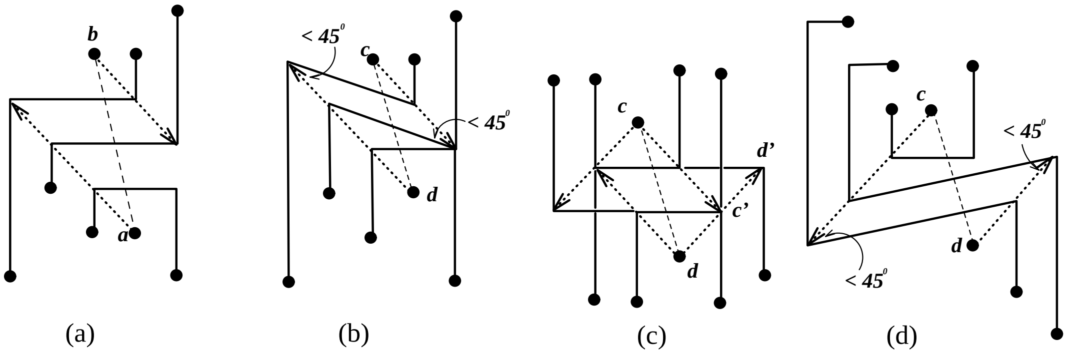


Figure 9: An unsafe cut either has an illegal vertex or has intersecting paths

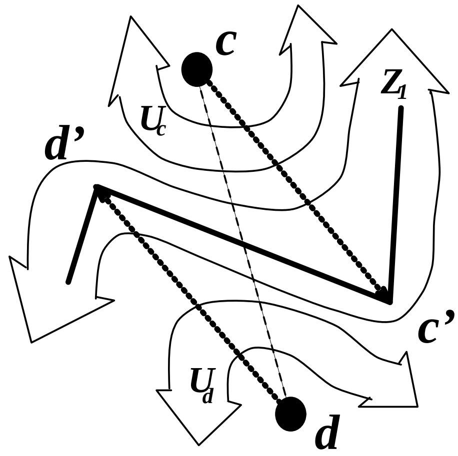


Figure 10: Illustration of an unsafe critical cut

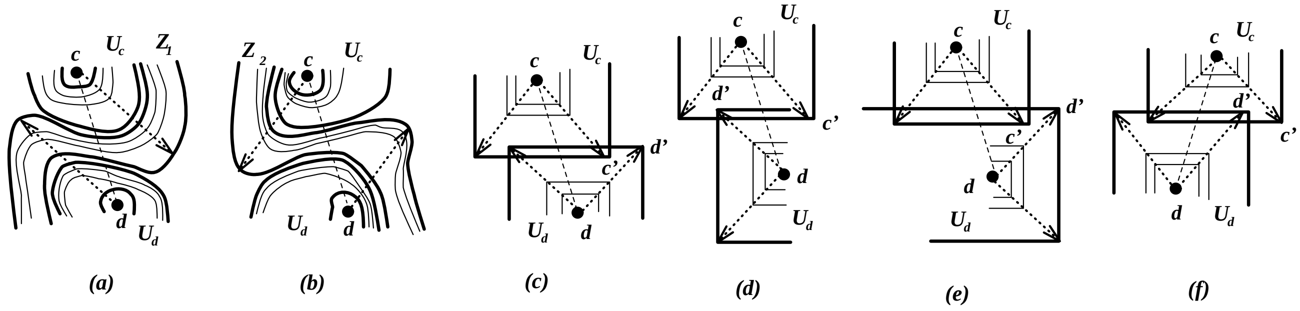


Figure 11: Characterization of wires crossing critical cut $\overline{cd}$

Theorem 2 *In a given sketch, an unsafe critical cut exists if and only if there exists an illegal vertex on some net* $\mathcal{N}^j \in W''$ *or there exist intersecting paths in* W''. (Fig 9)

Proof: Suppose critical cut $\overline{cd}$ with respect to feature c and d is unsafe. Without loss of generality, we assume $y_c > y_d$, and the rectangle defined by points c and d, as shown in Fig. 10 does not contain any other feature. If $y_c = y_d$ we find a feature d' such that $y_{d'} < y_d$ and the critical cut $\overline{cd'}$ is unsafe and it satisfies the above condition. If the rectangle defined by c and d contains other feature, there must exist an unsafe critical cut defined by features in $\{c, d\}$ and the rectangle, such that the above condition is true. The wires that cross $\overline{cd}$ can be classified into three categories, as is shown in Fig 11:

1. *Z-shape wires* supported by diagonal cuts from features c and d from top right to bottom left, denoted as Z_1. (Fig 11(a))
2. *Z-shape wires* supported by diagonal cuts from features c and d from top left to bottom right, denoted as Z_2. (Fig 11(b))
3. *U-shape wires* supported by at least two diagonal cuts emanating from either feature c or feature d, denoted as U_c and U_d.(Fig 11(c) to (f))

Note that $|Z_1||Z_2| = 0$, where $|Z_i|$, $i = 1, 2$, denotes the number of wires in Z_i, i.e., at least one of them is zero due to planarity property of the sketch. We distinguish three cases:

case 1: $|Z_2| = 0$ and $|Z_1| \neq 0$

Since critical cut $\overline{cd}$ is unsafe,

$$den(\overline{cd}) = |U_c| + |Z_1| + |U_d| > cap(\overline{cd}) = y_c - y_d - 1$$

Consider the supporting spoke $\overline{cc'}$ of some net $\mathcal{N}^j$ along diagonal cut $f_c(7\pi/4)$, where net $\mathcal{N}^j$ is the k^{th} net from feature c, i.e., $k = |U_c| + |Z_1|$. (Fig. 10) Let $\overline{dd'}$ be the supporting spoke of net $\mathcal{N}^j$ along diagonal cut $f_d(3\pi/4)$. We have

$$|U_c| + |Z_1| = y_c - y_{c'}, \quad |U_d| = (y_{d'} - y_d) - 1$$

That is,

$$(y_c - y_{c'}) + (y_{d'} - y_d) > y_c - y_d$$

or

$$y_{d'} > y_{c'}$$

which implies that both of the angles $\angle cc'd'$ and $\angle c'd'd$ are less than $\pi/4$. Thus, vertices c' and d' on net $\mathcal{N}^j$ are illegal.

case 2: $|Z_2| \neq 0$ and $|Z_1| = 0$

This case can be proved symmetrically.

case 3: $|Z_2| = 0$ and $|Z_1| = 0$ (See Fig 11 (c), (d), (e), and (f).)

Since critical cut $\overline{cd}$ is unsafe,

$$den(\overline{cd}) = |U_c| + |U_d| > cap(\overline{cd}) = y_c - y_d - 1$$

Consider the *outermost* nets $\mathcal{N}^j$ and $\mathcal{N}^k$ whose supporting spokes are $\overline{cc'}$ and $\overline{dd'}$ respectively. Since $|U_c| = y_c - y_{c'}$ and $|U_d| = y_{d'} - y_d$, we have

$$y_{d'} > y_{c'} - 1$$

The above inequality implies nets $\mathcal{N}^j$ and $\mathcal{N}^k$ either overlap or cross each other.

□

All the unsafe conditions are pictorially shown in Fig. 9, where (a) is safe, but (b) to (d) are unsafe conditions. We thus can devise our routability testing algorithm as follows. As mentioned earlier, it is necessary that the length of each obstructing spoke $f_i(\theta)$, not exceeding the length of the corresponding diagonal cut. (Otherwise, the capacity of the diagonal cut is exceeded by its density.) Suppose the spokes matrices pass the necessity test. After W'' has been computed, we need to scan for each net $\mathcal{N}^j$ in W'' to see if there exists any *illegal* vertex. This step only takes time proportional to the size of W'', i.e., $O(|F| \cdot |W|)$. From Theorem 2 we need to test if there exists any overlapping or crossing among all nets $\mathcal{N}^j$ in W''. Note that overlapping or crossing occurs only among U-shape wires. A U-shape wire with respect to a feature f_i is supported by at least two out of the four possible diagonal cuts

$$max_j(Matrix_{l\pi/4}(i,j) = Matrix_{((l+2) mod 8)\pi/4}(i,j)),$$

for some net $\mathcal{N}^j$ and $l = 1, 3, 5$, and 7.

We propose the following strategy. It is sufficient to keep for each feature f_i the outermost U-shape wire for intersection-testing purpose. See Fig. 12 for the four possible cases of U-shape wire around a feature. Thus each point feature generates at most one U-shape wire, and the intersection of these U-shape segments can be detected in $O(|F| \log |F|)$ time[15]. Extracting the outermost U-shape wire from the spokes matrices takes $O(|F| \cdot |W|)$. We conclude

Theorem 3 *The routability testing of a planar VLSI layout can be done in* $O(|F| \cdot |W|)$ *time.*

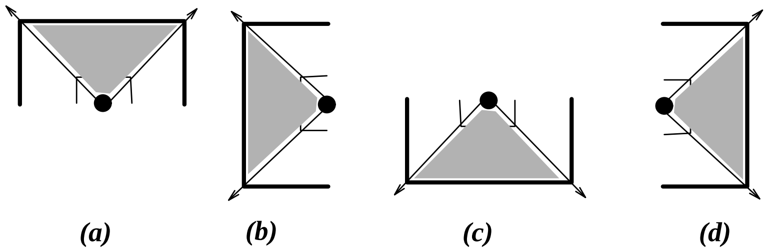


Figure 12: (a)$[\pi/4, 3\pi/4]$ (b)$[3\pi/4, 5\pi/4]$ (c)$[5\pi/4, 7\pi/4]$ (d)$[7\pi/4, \pi/4]$

3.6 Solving *DRH* Problem

To solve the *DRH* problem, we perform an additional phase to convert wire segments in W'' with slope other than 0 and ∞ into sequences of horizontal and vertical segments. Let $\overline{ab}$ be such a wire segment on net $\mathcal{N}^j$, and assume that its slope is positive as is shown in Fig. 13(a). The two vertices a and b define a rectangular region, called *routing window*, in which there may exist several negative spokes whose supporting vertices lie in the interior of the routing window. These spokes can be extracted from the spokes matrices when traversing the original net representation μ^j between points a' and b'. (i.e., Recall that $\overline{ab}$ is part of the shortest path after tightening the homotopy in the sleeve and those spokes of negative slope are exactly the boundaries of the extracted sleeve when we compute the vertex-disjoint RBE.) (Note that points a' and b' of μ^j have been transformed to a and b of $\mathcal{N}^j$ respectively.) The segment $\overline{ab}$ can be replaced by a *staircase* shape consisting of a sequence of vertical and horizontal line segments where only the spokes whose supporting vertices define a *maximal* vector are relevant, as is shown in Fig. 13(b). Similarly wire segments with negative slope are converted to a staircase shape supported by positive spokes within the routing window.

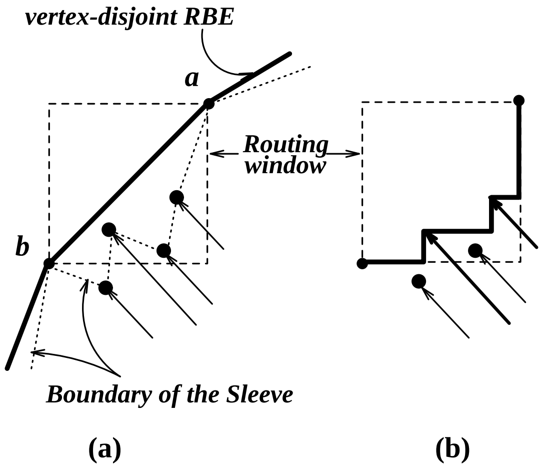


Figure 13: Converting a non-rectilinear wire segment (a) to a staircase (b)

Theorem 4 *The detailed routing given a homotopy problem (DRH), if the solution exists, can be solved in* $O(|F|\cdot|W|+k)$ *time, where* $k = O(|F|\cdot|W|)$ *denotes the total number of routing (horizontal* *and vertical) segments.*

4 Conclusion

We have presented an asymptotically optimal algorithm to find a vertex-disjoint RBE representation homotopic to the given sketch of a planar $VLSI$ layout.

Though the algorithm is presented on a grid model and all the minimal spacing between features is assumed to be one, it is not difficult to generalize the algorithm to the gridless model. Generalization to multi-terminal nets can be done following the idea of Kaufmann *et al.*[8] as described below.

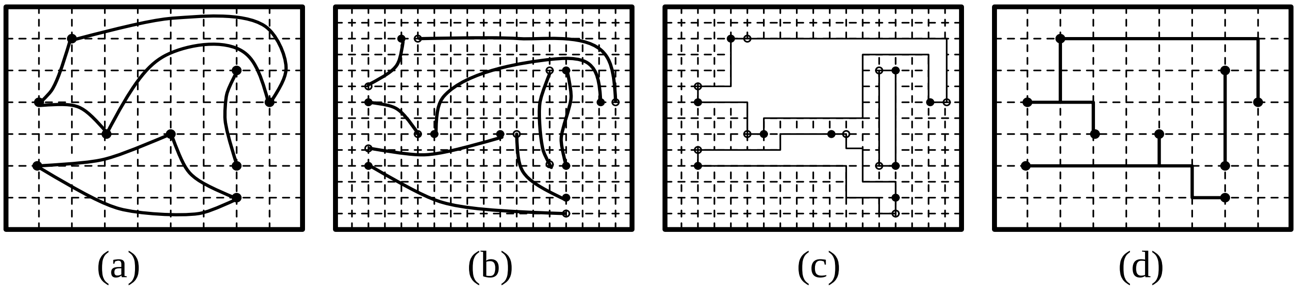
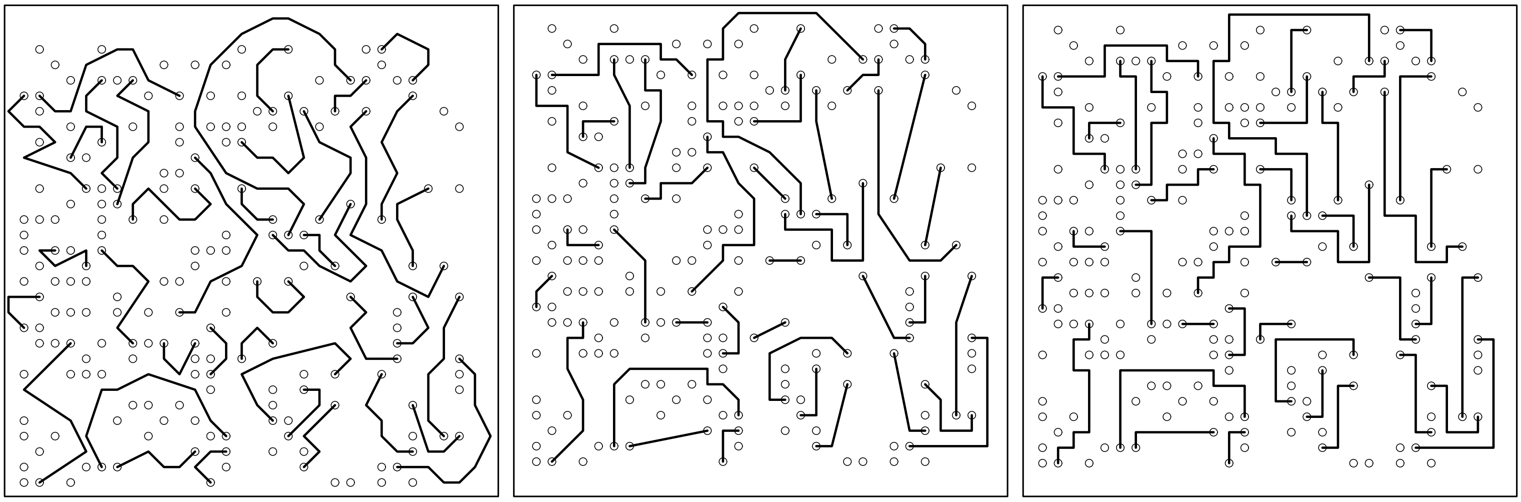


Figure 14: Kaufmann *et al.*'s multi-terminal net transformation.

For each s-terminal net ($s > 2$) we decompose it into s two-terminal nets by following the input multi-terminal net homotopy connecting two *adjacent* terminals at a time. (Two terminals in a multi-terminal net are said to be adjacent, if one can be reached from the other following *consistently* a clockwise (or counterclockwise) traversal at each Steiner junction.) Fig. 14(a) illustrates this idea. (In [8], an additional transformation step is performed by splitting each terminal into two neighboring terminals, as shown in Fig. 14(b).) We can then apply our proposed algorithm to transform each two-terminal net into its VRBE form. In general each s-terminal net, after the VRBE transformation, will produce a rectilinear polygon. Fig. 14(c) shows the output of Fig. 14(b) after the VRBE transformation. Each such polygon defines a routing region for the s terminals, and the algorithm for computing the Steiner minimal tree in a grid graph of Kaufmann *et al.*[8] can be applied. Fig. 14(d) shows the final result. The Steiner minimal tree algorithm[8] runs in $O(s^2 \cdot min\{s \log s, v\})$ time, where s is the total number of terminals of the multi-terminal net and v is the number of grid points inside the interior of the rectilinear polygon.

Finally we have assumed that the wire width is zero. In practice, the wire width is nonzero. To handle this case, consider the final routing to be along the middle of each wire segment, and the spacing between two wire segments will be the minimum spacing requirement plus the width of the wire, and the spacing between a wire and a feature will be the minimum spacing plus one half of the width of the wire.

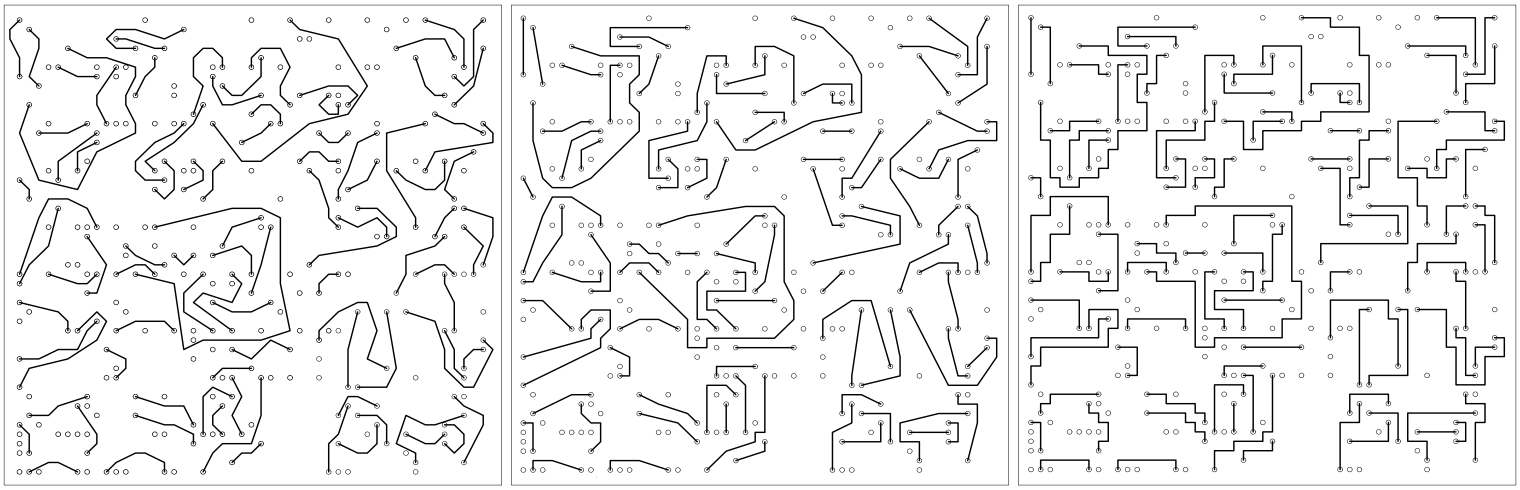
The algorithm can be used as a general tool for performing homotopic transformation of any planar layout. For example, it can be applied to remove the redundant U-shape routing and obtain an optimal layout[11][7]. The algorithm was implemented in C^{++} on a geometric input/output device, called GeoSheet, which is part of a geometric object manipulation and monitoring system developed at the Northwestern University. Fig. 15 and 16 show the computer output of two test examples that have the same topology as in Fig. 14 of [17], and in which the first diagram gives the input sketch, the second one gives its vertex-disjoint RBE transformation, and the third is the detailed routing. In Fig. 17, we show another example with longer nets and a slightly more complicated homotopy and its output.



(a) (b) (c)

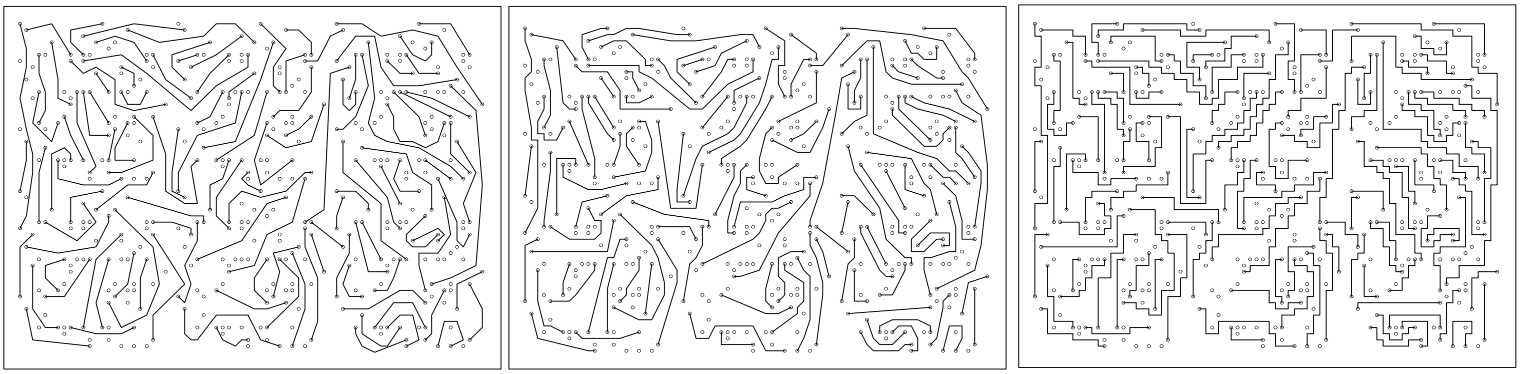
Figure 15: Test example S_6

Table I shows cpu time (SPARC 10) of some test examples of various input sizes. It seems to indicate that the running time is roughly propotional to the product of $|F|$ and $|W|$.



(a) (b) (c)

Figure 16: Test example S_9



(a) (b) (c)

Figure 17: Test example S_{10}

sketch	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
$\|F\|$	32	72	72	72	72	196	301	301	301	444
$\|W\|$	44	23	40	44	111	229	150	313	461	512
# of nets	4	1	2	3	20	39	35	67	97	124
$\|F\| \cdot \|W\|$	1408	1656	2880	3168	7992	44844	45150	94213	138761	2070528
cpu time (sec.)	0.3	0.3	0.3	0.5	0.9	2.3	2.5	4.4	5.2	6.4

Table 1: Test results

References

[1] R. Cole and A. Siegel, "River routing every which way, but loose,", in *Proc. 25th Ann. Symp. on Foundations of Computer Science*, pp.65-73, October 1984.

[2] W. W.-M Dai, R. Kong, and J. Jue, "Rubber Band Routing and Dynamic Data Representation", in *Proc. IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, pp.52-55, November 1990.

[3] W. W.-M. Dai, T. Dayan and D. Staepelaere, "Topological Routing in SURF: Generating a Rubber-Band Sketch", in *Proc. of the 28th ACM/IEEE Design Automation Conf.*, pp.39-44, 1991.

[4] W. W.-M Dai, R. Kong, and M. Sato, "Routability of a Rubber-Band Sketch", *Proc. of the 28th ACM/IEEE Design Automation Conf.*, pp.45-48, 1991.

[5] S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb, "On Continuous Homotopic One Layer Routing", in *Proc. of the 4th Annual Symposium on Computational Geometry*, pp. 392-402, 1988.

[6] S. Gao and F. M. Maley, "Advances in Homotopic Layout Compaction", in *Proc. of the 1989 Symposium on Parallel Algorithms and Architectures*, pp. 273-282, 1989.

[7] J.-M. Ho, A. Suzuki, and M. Sarrafzadeh, "An Exact Algorithm for Single-Layer Wire Length Minimization", *IEEE Trans. Computer-Aided Design*, vol. CAD-12, no. 1 pp. 175-180, January 1993.

[8] M. Kaufmann, S. Gao, and K. Thulasiraman, "On Steiner minimal trees in grid graphs and its application to VLSI routing", in *Proc. 5th International Symposium on Algorithms and Computation,* , Beijing, China, pp.351-359, August 1994.

[9] D. T. Lee and F. P. Preparata, "Euclidean Shortest Paths in the Presence of Rectilinear Barriers", *Networks*, Vol. 14, pp.393-410, 1984.

[10] C. E. Leiserson and F. M. Maley, "Algorithms for routing and testing routability of planar VLSI layouts," in *Proc. 17th Annu. ACM Symp. on Theory of Computing*, pp.69-78, 1985.

[11] K.F. Liao and M. Sarrafzadeh, "Boundary Single Layer Routing with Movable Terminals", *IEEE Trans. Computer- Aided Design*, vol. CAD-10, no.11, pp. 1382-1391, November 1991.

[12] F. M. Maley, "A Generic Algorithm for One- Dimensional Homotopic Compaction", *Algorithmica*, vol.6, pp.103-128, 1991.

[13] K. Mehlhorn and S. Näher, "A Faster Compaction Algorithm with Automatic Jog Insertion", *IEEE Trans. Computer-Aided Design*, vol. CAD-9, pp.451-466, February 1990.

[14] R. Y. Pinter, "River Routing: methodology and Analysis", *Proc. 3rd Caltech Conf. on VLSI*, pp.141-164, 1982.

[15] F. Preparata and M. Shamos, Computational Geometry, An introduction, Springer-Verlag, 1985.

[16] D. Richards, "Complexity of Single-Layer Routing", *IEEE Trans. on Computers*, vol. C-33, pp. 286-288, March 1984.

[17] M. Sarrafzadeh, K.-F. Liao, and C. K. Wong, "Single-Layer Global Routing", *IEEE Trans. Computer-Aided Design*, vol CAD-13, pp. 38-47, January 1994.

figure captions

Fig. 1. (a) Layout specification, (b) rubber-band equivalent and (c) detailed routing.

Fig. 2. (a) A topological layout instance and (b) its vertex-disjoint RBE.

Fig. 3. Illustration of pockets and the nesting property.

Fig. 4. Removing fictitious intersections.

Fig. 5. Introducing a pseudo net.

Fig. 6. A sleeve in a triangulated simple polygon.

Fig. 7. Boundary of a sleeve formed by obstructing spokes.

Fig. 8. Transformation to W''_+ and W''_-.

Fig. 9. An unsafe cut either has an illegal vertex or has intersecting paths.

Fig. 10. Illustration of an unsafe critical cut.

Fig. 11. Characterization of wires crossing critical cut $\overline{cd}$.

Fig. 12. (a)$[\pi/4, 3\pi/4]$ (b)$[3\pi/4, 5\pi/4]$ (c)$[5\pi/4, 7\pi/4]$ (d)$[7\pi/4, \pi/4]$.

Fig. 13. Converting a non-rectilinear wire segment (a) to a staircase (b).

Fig. 14. Kaufmann *et al.*'s multi-terminal net transformation.

Fig. 15. Test example S_6.

Fig. 16. Test example S_9.

Fig. 17. Test example S_{10}.

TABLE I Test Results