Fast and Incremental Routability Check of A Topological Routing Using a Cut-based Encoding

Man-Fai Yu
Wayne Wei-Ming Dai

UCSC-CRL-97-07
April 14, 1997

Baskin Center for
Computer Engineering & Computer Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

Many performance-driven routing algorithms do not consider routability. Routing trees are built assuming that there are no other wires. The main reason for this is that it is NP-hard to guarantee routability. Even checking for routability is a time-consuming process. This limits the usefulness of many performance-driven routing algorithms because unroutable designs are useless. Previous online routability checking is not fast enough for the many iterative improvement steps in a performance-driven routing algorithm. This paper provides such an algorithm.

We propose a versatile topological routing encoding that not only allows an efficient routability check, but also provides proximity information for crosstalk and manufacturability analysis. Our routing model is applicable to a wide range of technologies, including PCB, MCM and standard cell ASIC. It allows rectilinear, octilinear or Euclidean wiring metric and arbitrary-shaped obstacles. It is gridless and supports variable wire width and spacing. Our algorithm is fast enough to be integrated into any iterative improvement schemes such as simulated annealing. Our basic observation is that the placement of obstacles is fixed but rerouting is very frequent. So we emphasize on an efficient rerouting and routability check step but pushes the complexity to building a data structure that depends only on the placement of the obstacles.

Keywords: topological routing, planar routing, routability, single-layer routing, design rule check, visibility graph

1 Introduction

We propose an efficient and versatile encoding scheme of topological routing for fast and incremental routability checking. This enables us to do online checking for automatic iterative improvement such as simulated annealing. Our design rule checking is based on cuts and flows. It is gridless, supports variable wire width and spacing, allows rectilinear, octilinear (45 degree) or Euclidean wiring metrics and can have obstacles of arbitrary shape.

Many traditional design rule checkers are grid based and restricts all geometry to rectilinear. In a high-performance system, we adjust wire widths for optimal delay and wire spacing for crosstalk control. In modern ASIC design there are large irregular macro blocks mixed with small standard cells with over-the-cell routing. This technology shift calls for general area routing.

Our design rule checking system is based on *topological routing*. A topological routing is an embedding of wires connecting all terminals as specified by a netlist but not guaranteed to be design-rule correct. The shape of a wire can be changed but a wire is not allowed to cross another wire or an obstacle. Generating a routable topological routing is known to be NP-hard.

The motivation that drives this research is the need for a fast and incremental routability checker. For large designs, changes to a routing are inevitable. These changes can come from either the designers for last-minute bug fixes or from routing-improvement tools such as wire-sizing or buffer-insertion. The router has to 1) allow the changes to be done quickly because a tool may generate many rip-up and reroutes, 2) decide that whether the new routing is routable or not. The only viable way to do this is by devising an incremental algorithm. We achieve these two objectives by noting the following. First, maintaining topological routing takes less effort than the full shape and size of each wire in detail. Second, routability is completely determined by topological routing.

The Surf system allows the user to directly manipulate topological routings[1]. Valainis et al[2] reported a compaction system using topological layout. Since the final geometry of the wires are not fixed, many performance-oriented optimizations can be applied. For example, wire widths and spacings can be adjusted for yield improvement, impedance matching, crosstalk control and delay. Wire lengths can be adjusted for timing and skew constraints. By not explicitly recording the exact geometry of all the wires, the amount of information that needs to be changed during rip-up and reroute is less. This leads to efficient schemes of iterative improvement of topological routing.

We first describe our routing model in Section 2. Then we give a brief summary of the major properties of wires and cuts. The majority of the results is from Maley[3]. Then we give an estimation of the number of critical cuts. This is the number of cuts we have

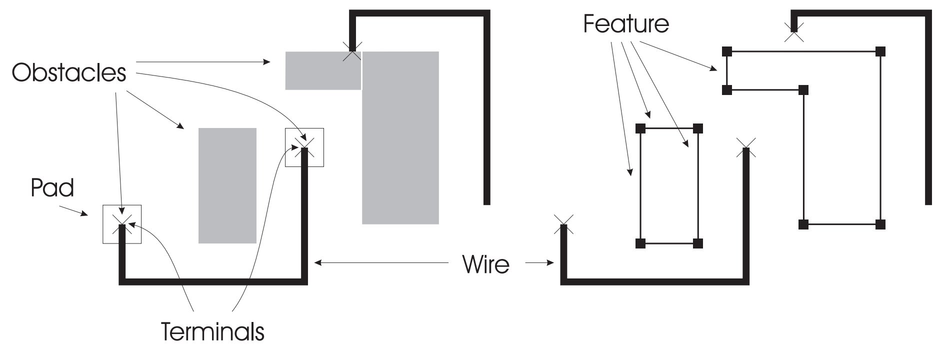

Figure 1: The left figure shows our routing model. There are obstacles, wires and terminals. Wires terminate at terminals. Terminals can be standalone or on the edge of an obstacle. A terminal is also an obstacle. An obstacle is a point, a line segment or a convex simple polygon. The right figure shows Maley's routing model. Features are straight line segments. Obstacles are composed of many features. Terminals are not allowed on an obstacle.

to check for routability. The detail derivation is deferred to Appendix A. We showed that the number of critical cuts is asymptotically smaller under rectilinear wiring metric than under Euclidean metric. Section 5 and 6 are the core part of this paper. We present our *cut-based* encoding for topological routing. Cuts with the same flow are grouped into *cut classes* which represent topological relationships between obstacles. The most difficult part of a cut-based encoding is to compute the *topological encoding graph.* This graph depends solely on the relative placement of obstacles and terminals. Our efforts paid off in checking design rules for a particular routing among these obstacles as shown in Section 7. Since obstacle placement is done only once but a routing is iteratively changed, our strategy is efficient overall.

2 The Routing Model

The routing model is an abstraction of a real-world routing problem. It should be sufficiently simple to allow theoretical development but real enough for the developed ideas to be useful in the real world. Our model is based on printed-circuit board (PCB), multichip module (MCM) and ASIC standard cell technology. In these processes, there are a number of components (or cells) with I/O pins. An I/O pin is in the form of a metal pad. There are fixed *obstacles* representing keep-outs, blockages, drill holes or intracell routing. This model has a clear distinction between wires and obstacles. It is not suitable for cell-level routing where flexible polygons are the primary objects. Nor is it suitable for microwave circuits where the precise geometry of the interconnect is required. To simplify formulation, we assume that all geometries are piecewise-linear. For technical reasons which we will explain below, we require that all obstacles are convex. Our model allows concave obstacles to be represented as a group of abutting convex obstacles so there is no practical limitation on the shape of obstacles.

On a grid-based layout system, obstacles are defined by clusters of grids. Terminals are also on grids[4, 5]. These systems need many grid points to represent large obstacles. This is because their grid points is a uniform discretization of the routing space. We directly supports obstacles as objects and does not restrict them to be on a grid. This is important in performance-driven routing systems because wires may be sized for delay and spacing may be irregular for crosstalk control. In Maley's work[3], the routing model is gridless but terminals are not allowed to touch an obstacle. This is to avoid the technical difficulty in dealing with apparent design rule violation when a wire connecting to a terminal on the boundary of an obstacle gets below the minimum spacing with respect to the obstacle. In real designs, it is possible that an obstacle represents a macro block and its pins are on the boundary of the block. Our routing model supports this by attached terminals. We will discuss this further after we have defined the topological encoding graph(Section 5).

Fig. 1 shows the objects in our routing model. A *terminal* is a point where a wire ends. A terminal can be standalone or on the boundary of an obstacle. An *obstacle* is a simple polygon. In PCBs or MCMs, wires usually end at a pad. In our model, a pad is represented by a single point—a terminal. The dimensions of the pad are taken into account by reducing the capacity of the cuts ending on this terminal. More than one wire may end on one terminal. If a terminal is on the boundary of an obstacle, we say it is *attached* to the obstacle. Otherwise it is a *standalone* terminal.

The routing boundary P is a simple polygon that contain all the routing objects. The *routing region* $B = P - X - T$ where X is the set of all obstacles and T the set of all terminals. B is usually an open multiconnected space.

3 The Routability Theorem

This section briefly introduces some result of Maley[3] on which we develop the encoding. The most important concepts are cuts and flows. Intuitively, we measure the wires between a pair of obstacles. If the total width and spacing used by the wires is less than the spacing between these two obstacles, then these wires can be routed successfully between them. The path we use to measure between two obstacles is called a cut. The total amount of space used by the wires (with spacing) is the flow of the cut. The total amount of space that can be used to route wires is the capacity of the cut. The flow has to be always less than the capacity for all cuts in order for the whole design to be routable.

A major result of Maley[3, esp. Ch. 2] is that we only need to consider the shortest cuts and wires. To decide whether a topological routing is routable, we first compute the rubberband equivalent of all the wires. A rubberband equivalent (RBE) of a wire is the shortest path that is *path homotopic* to the wire. Further, the topology completely determines routability.

A *path* p is a continuous piecewise linear function $p : \mathbf{I} \to B$ where $\mathbf{I}$ is the interval [0..1] and B is the routing region. A wire is a path that does not touch any other wires or obstacles except at the terminals. The *endpoints* of a path p are the points $\{p(0), p(1)\}$. A path p is *path homotopic* to a path q if there exists a continuous piecewise-linear function $H : \mathbf{I} \times \mathbf{I} \to B$ where $H(0, \cdot) = p$ and $H(1, \cdot) = q$ and their terminals coincide.

Maley used the concept of *features* instead of obstacles (Fig. 1). A feature is a single point or a straight line segment that touches nothing except at its endpoints. Hence an obstacle is a collection of features. The use of features simplifies a lot of mathematical development for routability because straight line segments are convex and simple to describe. The downside is that a simple polygon is broken down into many straight line segments so the layout system has to keep track of many objects. In Section 5, we see that many cuts between features are redundant because these features (straight line segments) are part of the boundary of a single obstacle. So although features are convenient for mathematical treatment, we use obstacles as our basic objects. This is one of the reasons why our encoding is compact.

A cut is a path that starts and ends on features. A cut is *between* two features if it starts and ends on the two features. The *flow* of a cut is the sum of the width and spacing of all the RBEs of wires that intersect the cut. The *capacity* of a cut x, denoted $\operatorname{cap} x$, is the maximum total width plus spacing wires can cross without violating design rules.

Theorem 1 (Maley's Sketch Routability Theorem) *A topological routing is routable if and only if all shortest straight cuts between all pairs of visible features are safe.*

A cut is *safe* if the flow of the cut is less than or equal to its capacity. A feature f is *visible* from a feature g if there exists a pair of points p and q on f and g respectively such that a straight line can be drawn from p to q without touching or intersecting any other features. The set of all cuts that needs to be checked for safety is called the *critical cut set.*

By Theorem 1, the set of all cuts is the set of edges of the visibility graph with endpoints of features as vertices. It is natural to assume that the number of edges of this graph depends heavily on the routing metric and the placement of features. The following section gives quantitative answers to this question.

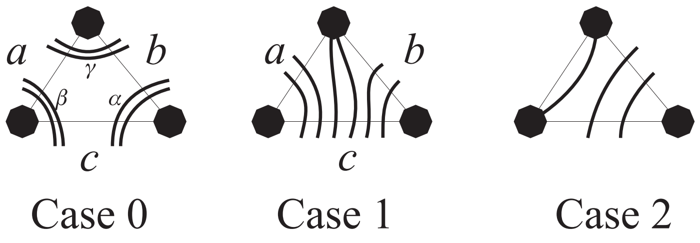

Figure 2: Three unique cases to derive a topological routing from numbers on the edges of a triangle.

4 Problem Size Estimation

According to Theorem 1, we have to check the shortest straight cuts between each visible pair of features. Hence the number of cuts to check is the number of edges in the visibility graph where the vertices are the features. The cost of design rule checking is directly proportional to the number of cuts we need to check. So it is in our interest to find the minimum number of cuts to check that is necessary to determine routability. It is clear that we have to check at least one cut per pair of visible features. The dominance relation described in Section 5 allows us to reduce the number of cuts to check between one pair of obstacles. We still have to check cuts between *different* pairs of obstacles. For m features, in the worst case we need to check $m(m-1)/2$ cuts. In most practical situations, the number of cuts to check is a lot less. Appendix A give detailed estimation on the size of the critical cut set in a uniform grid. The main results are:

- The number of cuts to check is $O(n^2)$ where n is the number of obstacles but the constant is smaller than the worst case.
- For octilinear wiring metric, the number of cuts to check for *each* grid point is $2(n-1)+1 \in O(n)$. Therefore it is $O(n^2)$ for all points.
- For rectilinear wiring metric, we only need to check 4 cuts for each grid point so the total number of cuts to check is only $O(n)$.

5 A Cut-based Topological Encoding

In almost all layout systems, wires are recorded as sequences of line segments. They can be collectively called *wire-based* encoding. For a given wire, this encoding can report quickly the shape, width, length and exact geometry of the wire. This encoding is more awkward when *proximity* information is required. Given a wire, it is not easy to identify at any given point the nearest neighboring wire. This information is important for routability and crosstalk computation.

In this paper, we propose a *cut-based* encoding. In this encoding, the main objects in the encoding are cuts. Each cut contains a sequence of intersecting wires. This approach was used by several researchers for PCB routing, compaction and layout optimization [8, 9, 10, 2, 11]. Our encoding is based on Yu, Darnauer and Dai[11]. Since any encoding necessarily describes the topological routing, we can create a corresponding wire-based encoding from a cut-based encoding and vice versa.

The fundamental advantage of cut-based encoding is that the basis of the encoding is detached from the routing. In our encoding the basis of the encoding is a triangulation based solely on the placement of obstacles and terminals. In a typical routing environment, the placement of these objects are fixed whereas the routing changes frequently. We record the routing as a simple vector of numbers on top of the basis. Changing the routing is therefore very easy. The cuts carries information between adjacent wires and obstacles. In this paper we take advantage of this information by building an efficient routability test (Section 7). Other performance constraints that relates to proximity information such as crosstalk control, defect analysis or electromigration can also take advantage of such information.

Our encoding is based on a discretization of the routing space with triangles. The Triangle Encoding Theorem (Theorem 2) establishes our base—the topology of each triangle can be determined easily. The major part of the development is on the *dominance* relationship between cuts. Due to dominance, cuts can be grouped into equivalence classes where it is sufficient to use one cut to represent the whole class (the dominant cut). This development culminates to the introduction of the *topological encoding graph* (TEG). The edges of this graph represents the *necessary* critical cuts, i.e., routability cannot be determined if one cut is missing from this graph. A TEG in general is not planar. The most important result of this section is that a TEG always contain a triangulation (Lemma 6). Using (one of) the triangulation as the basis, we introduce our cut-based encoding. The cuts that are part of this triangulation are *explicit* cuts. An explicit cut is both a cut (that needs to be checked) and a vehicle for encoding a topological routing.

We first assume that all wires are of the same width and require the same minimum spacing. Later in this section we suggest ways to extend this to support wires of uneven width and different requirements of minimum spacing.

Our basic observation is the following:

Theorem 2 (Triangle Encoding Theorem) *Given three cuts forming a triangle and the flow of each cut, we can decide, for the area within the triangle, either*

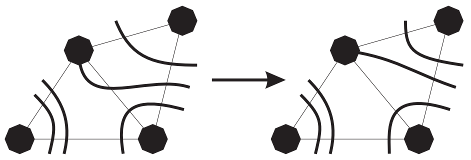
Figure 3: A wire connecting to a vertex should not intersect a cut incident to it. It is an unnecessary crossing and it can be represented by one without the crossing.

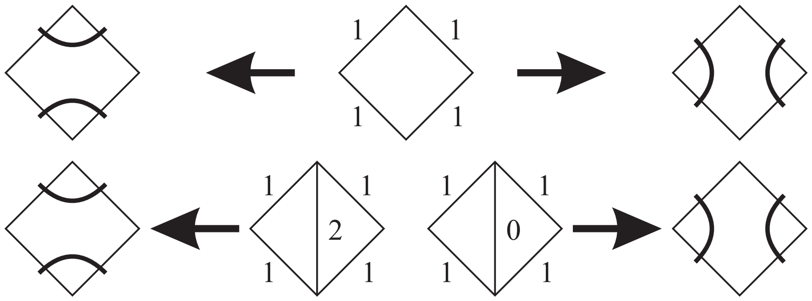

Figure 4: The topological routing is not unique if flows are only specified on a polygon with more than 3 sides.

1. *no topological routing exists, or*
2. *there exist a unique topological routing.*

Proof: Refer to Fig. 2. Only three cases are possible within a triangle. In the first case (Case 0), there are no wires connected to any vertex. In this case, consider the following equations,

$$(b+c-a)/2 = \alpha \quad (c+a-b)/2 = \beta \quad (a+b-c)/2 = \gamma$$

Given (a, b, c), either (α, β, γ) has a positive integer solution or not. If so, the solution is unique. If the solution exists, the topological routing can be constructed as shown in the figure. If not, a topological routing does not exist.

If there is one or more wires (Case 1) connecting to a vertex, it must intersect the cut not adjacent to the vertex. Refer to Fig. 3. In this case a valid unique topology exists if $c > a + b$.

Finally, a unique topology exists for a triangle with two connections (Case 2) if $a = 0$ and $b = c$ or their symmetric permutation thereof. The only difference with Case 0 is that there is a wire on top of a wire. This can be checked from the given netlist. $\square$

Fig. 4 shows that topology can only be uniquely specified with a triangulation. However, in a quadrilateral if one of the cuts has zero flow, the topology can be specified using

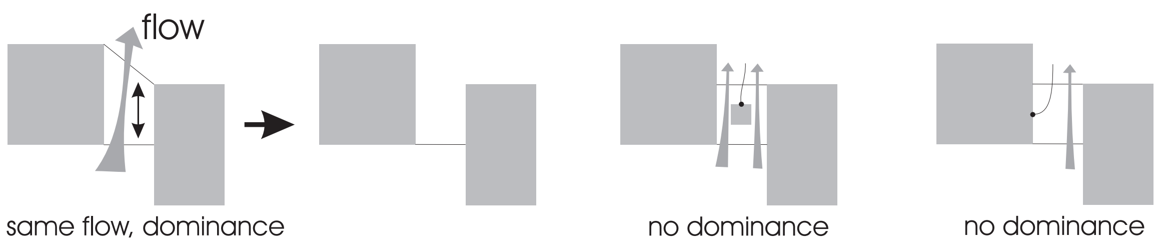

Figure 5: Reduce the critical cut set using Corollaries 3 and 4. This is because we know that the flow across the edge of an obstacle is zero. In general, we only need to check a few cuts between any pair of convex polygons.

Theorem 2 by simply collapsing the empty cut. In general, we have the following corollaries.

Corollary 3 *Routing topology can be decided within an n-sided polygon if the flow is zero on at least $n-3$ edges.*

Corollary 4 *Given an n-sided polygon, let cuts a and b be the only two cuts with non-zero flow. The flow of a is equal to the flow of b if there is no connection within the polygon. If* $\operatorname{cap} a \leq \operatorname{cap} b$*, we say a* dominates *b.*

a dominates b because they have the same flow so if the lower capacity cut is safe, the other cut is safe. We can avoid checking many cuts using Corollaries 3 and 4 (Fig. 5).

The dominance relationship gives us a powerful way to reduce the number of cuts we need to check for routability. Intuitively, dominance can only happen between cuts that are between the same pair of obstacles. However, not any pair of cuts between a pair of obstacles have a dominance relationship. This is because there may be another obstacle (Fig. 5). The concept of *boundary homotopy* formally captures this situation.

To simplify the treatment of attached terminals, we consider an attached terminal a standalone terminal infinitesimally close to its obstacle. This is simply represented by linking them with a cut with zero capacity.

A path q is *boundary homotopic* to a path p if there exists a piece-wise linear continuous function $H : \mathbf{I} \times \mathbf{I} \rightarrow B$ where $H(0, \cdot) = p$, $H(1, \cdot) = q$, $H(\cdot, 0) \in b$ and $H(\cdot, 1) \in c$ for some distinct obstacle boundaries b and c. b or c can be a point if an end point of q or p is on a standalone terminal. Since all obstacles are convex, we do not have to check cuts that end on the same obstacle.

It is easy to verify that boundary homotopy is an equivalence relation. If two simple cuts p and q are boundary homotopic with the homotopy H, then the area bounded by p, q, $H(\cdot, 0)$ and $H(\cdot, 1)$ has no standalone terminals or obstacles. Therefore either the two cuts both intersect a wire or they both do not because wires end at terminals. By Corollary 4, two boundary homotopic cuts have the same flow. So all cuts within an equivalence class (*cut class* for short) has the same flow. The cut with the lowest capacity is called the *dominant* cut of the cut class.

Maley proved that all critical cuts are straight[3]. From now on we will only consider straight cuts. We say that two cut classes π_1 and π_2 *intersect* if the endpoints of π_1 and π_2 are on different obstacles and for any straight cut in π_1 there is a straight cut in π_2 that intersects it. Note that the intersection concept does not apply to cut classes that share at least one obstacle. The intersection of cut classes captures the topological relationships between obstacles. This is observed by the following lemma.

Lemma 5 *Let the set of cut classes intersected by a cut class π be $I(\pi)$ and the set of two terminating obstacles $E(\pi)$. Two cut classes $\pi_1 = \pi_2$ if and only if $I(\pi_1) = I(\pi_2)$ and $E(\pi_1) = E(\pi_2)$.*

PROOF: The "only if" part is straightforward. If two cut classes are the same, they must terminate on the boundaries of the same set of obstacles. Also, the definition of intersection implies that if one cut in a class intersects some cut, then all cuts in the class intersects the same cut. Therefore the intersection counts are the same if the two classes are the same.

Now we consider the "if" part. Contrapositively, if $\pi_1 \neq \pi_2$, there are two cases. The first case is that their terminating obstacle boundaries are different, hence $E(\pi_1) \neq E(\pi_2)$. Now we consider the second case. We can assume that π_1 and π_2 terminates on the same obstacle boundaries. Let $p_1 \in \pi_1$ and $p_2 \in \pi_2$ be two cuts. Let b_1 be the (partial) boundary between $p_1(0)$ and $p_2(0)$ and b_2 be the boundary between $p_1(1)$ and $p_2(1)$. Since $\pi_1 \neq \pi_2$, there is an obstacle s such that the loop $b_1 \circ p_1 \circ b_2 \circ p_2$ is inessential.[1]

A straight cut c terminating at s intersects either p_1 or p_2 because b_1 and b_2 are part of obstacle boundaries. c cannot intersect both p_1 and p_2 because it is straight. There is at least one c because the whole routing region is bounded by the bounding polygon. Hence $I(\pi_1) \neq I(\pi_2)$. $\square$

This lemma says that a cut class is uniquely specified by where it is terminating at and what other cut classes it intersects. To record a cut class, all we need is the names of the obstacles it terminates at and the names of cut classes that intersects it. We will use this property to construct our encoding.

We are ready to define the *Topology Encoding Graph (TEG)*. This graph is the basis of our encoding scheme. For a routing instance, let the set of obstacles be X, the set of terminals be T, the set of dominant cuts from all cut classes in $T \cup X \cup \{0\}$ be C, where 0 is the routing boundary polygon. Let D be a maximal planar subset of C.

[1]that can be shrinked to a point.

Definition 1 (Topology Encoding Graph) *A Topology Encoding Graph of a routing instance is a graph with vertices defined as follows:*

- *T, the* terminal *vertices.*
- *X, the* obstacle *vertices.*
- *$\{0\}$, the boundary vertex, representing the routing boundary polygon.*

The edges are defined as follows.

- *An edge* (terminal-obstacle edge) *exists between an attached terminal and its obstacle.*
- *For each dominant cut c in D, there is an edge between its terminating obstacles/terminals.*

Each edge in a TEG has a capacity. The capacity of a dominant-cut edge is the capacity of the cut under the current wiring metric. For a terminal-obstacle edge, its capacity is permanently set to 0.

Since D is planar and no obstacles overlap, a TEG is planar. We call the dominant cuts that corresponds to an edge in a TEG *explicit* cut. Other cuts are *implicit* cuts.

Definition 2 (Cut-based Encoding) *A* Cut-based Encoding *of a topological routing is a triple (G, M, x) where G is a TEG of the routing instance, x is a vector of flows for each edge in T and M is the set of implicit cuts. Each implicit cut is represented by its beginning and end vertices and the sequence of its intersecting explicit cuts.*

Checking the safety of explicit cuts is trivial. Checking implicit cuts is not that hard either. Section 7 presents an efficient algorithm for checking implicit cuts. An encoding is proper if we can decide from the encoding that either 1) a topological routing does not exist, or 2) there is a unique topological routing and in the latter case, deduce the routing from the encoding. We will now show that our cut-based encoding is proper. Our main approach is to show that TEG is a triangulation so that for each triangle, we apply the Triangle Encoding Theorem.

We start with a proof that a maximal planar subgraph of a shadowed visibility graph is a triangulation.

Lemma 6 *A maximal planar subgraph of a shadowed visibility graph under the rectilinear metric is a triangulation.*

Proof: Suppose there is a maximal planar subgraph that is not a triangulation. Then there is a face with more than 3 vertices. Name these vertices in counterclockwise order

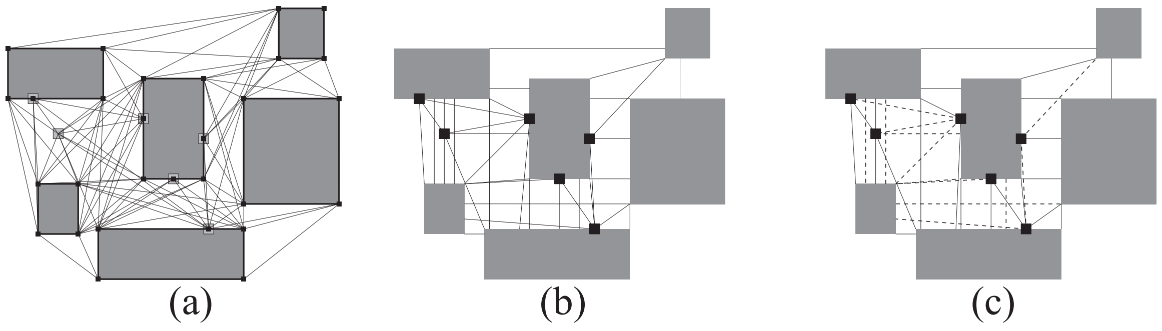

Figure 6: The critical cut set is greatly reduced after applying dominance relationship and Corollary 3. (a) shows the cuts that needs to be checked according to Maley[3]. (b) shows all the dominant cuts. It is sufficient to check these cuts. From these cuts, we choose a maximal planar subset as our encoding basis (c). These are the *explicit* cuts. Other cuts are *implicit* cuts.

$v_0, v_1, \ldots, v_{m-1}$ of m vertices. There is no edge between a vertex v_i and all other vertices except v_{i-1} and v_{i+1} so v_i is not 4-visible to all vertices except v_{i-1} and v_{i+1}. Refer to Fig. 16. For any i, v_i and v_{i+2} must be in the opposite quadrant of v_{i+1}. So v_{i+2} is in the shadow of v_{i+1}. The same argument holds for v_{i+1} so v_{i+3} is in the shadow of v_{i+2}, which is in the shadow of v_{i+1}. Repeating the argument, starting at v_0, $v_2, \ldots, v_{m-1}$ are in the shadow of v_1. This leads to a contradiction because v_{m-1} should be 4-visible from v_0. □

Since a shadowed visibility graph under rectilinear metric is a subgraph of one under octilinear metric, which is a subgraph of one under Euclidean metric, this lemma holds for octilinear and Euclidean metrics.

Consider the visibility graph of all obstacle boundaries broken down as straight line segments and terminals (Fig. 6). A maximal planar graph H of this visibility graph is a triangulation according to Lemma 6. Not all the edges in the visibility graph has a corresponding edge in the TEG because the TEG only records dominant cuts. However, no triangles will be added or removed because edges within the same cut class forms multiple edges between two vertices instead of triangles. From the above arguments, we showed that the cut-based encoding scheme is proper.

Our encoding is minimal because each cut is topologically significant. Removing any cut from an encoding we will miss a critical cut. The number of vertices is also minimal because each terminal or obstacle is represented as one and only one vertex. The most important concept behind our encoding is that the routing space is not discretized uniformly. It is discretized (by triangles) only if *it makes a topological difference.*

If wire width and spacing are non-uniform, we have to record the wires intersecting

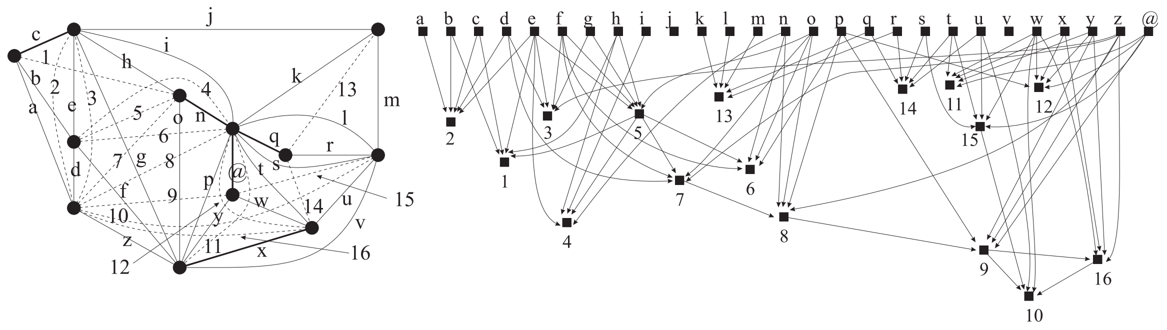

Figure 7: A TEG and its dependency graph. The explicit cuts are labeled as alphabets and implicit cuts as numbers. The left is a TEG of Fig. 6. The right is (one of) its loop-free dependency graph. Using the dependency graph, we can compute the flows of all implicit cuts very quickly (Section 7).

each cut in the TEG in order. Note that the ordering information is already embedded in the topological routing. Explicitly recording it makes safety checks much more efficient. Therefore the ordering information is a form of cache and can be regenerated by applying the Triangle Encoding Theorem to all triangles.

6 Computing An Encoding

Given a set of obstacles, the bounding polygon and a set of terminals, we wish to generate a cut-based encoding excluding the flow vector because in a cut-based encoding, all the routing information is captured by the flow vector. We want to create a TEG and a set of implicit cuts.

We start with a visibility graph of the obstacles. All critical cuts are edges of the visibility graph. Many edges in the visibility graph are redundant because they are dominated by other cuts. We eliminate them by computing the intersections of all pairs of cuts and remove those with identical intersections according to Lemma 5. We then choose some cuts to form a triangulation and leave the rest as implicit cuts. Fig. 6 shows our process.

We first represent all polygons as multiple line segments (*features* in Maley[3]'s terminology). For an obstacle with attached terminals, we break up its boundary at the attached terminals and at its polygonal vertices (Fig. 6a). We use Ghosh's algorithm[12] to compute the visibility graph between all pairs of line segments and/or points. The complexity of this algorithm is $O(n \log n + E)$ where n is the number of points and endpoints of line segments and E the number of edges in the visibility graph. If an edge exists between two line segments, the corresponding cut is the shortest straight cut between them. If there are

Algorithm 1 (Encoding)

```
Algorithm Encoding (Obstacles $S$, Bounding polygon $B$, Terminals $T$)
    Convert all the polygons to features $F$ from $S \cup B \cup T$
    Compute the visibility graph $G$ from $F$
    Compute intersections among cuts from $G$
    Remove redundant cuts so that each cut has a unique set of intersections.
    Pick enough cuts to form a triangulation, designate them as explicit cuts
    Designate the rest as implicit cuts
```

Figure 8: Algorithm Encoding

many cuts of the same capacity, we arbitrarily choose one.

Next we compute all the intersections of these cuts. We use an algorithm described in Preparata and Shamos[13, pp. 284] to compute all the intersections between all pairs of cuts in $O((K + E) \log E)$ time where K is the number of intersections. We have to remove all but one cut with the same terminating obstacles and intersections because they belong to the same cut class according to Lemma 5. This can be done in $O(E)$ time using a hash table with a key composed of the terminating obstacle names and the names of all intersecting cuts. For each cut class we save the dominant cut.

Now we can form a TEG by simply pick the cuts to form a triangulation. A simple $O(E)$ algorithm for this is to choose an unmarked cut, mark all the cuts that intersect this cut and repeat. All the remaining unmarked cuts are maximally planar because they do not intersect each other. By Lemma 6 we have a triangulation. Lemma 6 guarantees that a triangulation exists. The remaining cuts are implicit cuts. Fig. 8 summarizes the whole process.

7 A Fast Design Rule Check Algorithm

Given a topological routing encoded in a cut-based encoding, we want to decide its routability. By Theorem 1, we need to decide the safety of all the cuts. The flow of each explicit cut is known because that is part of the encoding so the safety of each explicit cut can be determined trivially. This section presents an efficient algorithm to determine the safety of implicit cuts.

Our basic observation is shown in Fig. 9. Given a pair of triangles that share one edge, we can find the flow of the diagonal implicit cut. Fig. 10 shows 6 different wire topology configurations with the same formula for the flow of f. Fig. 11 shows another two cases involving multiple fanouts. Any wire topology can be classified into one of these 8

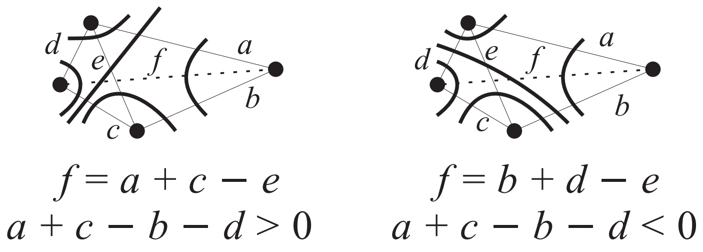

Figure 9: Computing the flow of an implicit cut f in a quadrilateral. There are two possible topological routings. The flow of the implicit cut, f, is different in these two cases. The inequality under each case distinguishes each other.

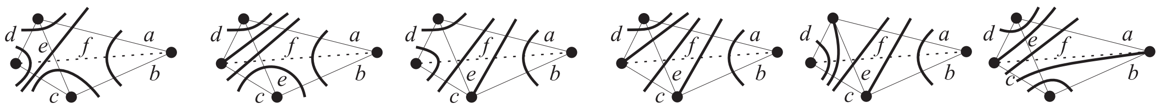

Figure 10: 6 topology configurations of two triangles and a diagonal implicit cut. In all cases, the flow of the implicit cut, f is equal to $a + c - e$ or $b + d - e$ depending on the polarity of $a + c - b - d$.

configurations with a suitable rotation and naming of edges. In short we have the following lemma.

Lemma 7 *Given two triangles sharing a cut, we can decide the flow of the diagonal implicit cut in constant time.*

Since the flow of an implicit cut can be computed from five cuts, we define a *dependency* graph to capture this relationship. From the dependency graph, we can compute the flow of any implicit cut.

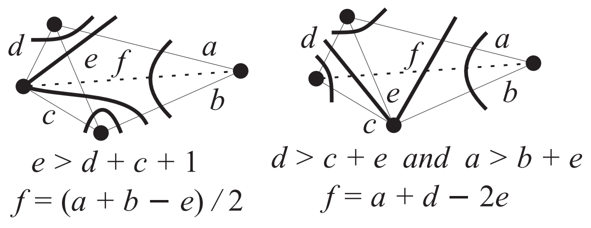

Figure 11: 2 topologies that need special handling. Both happens when there is more than one wire connected to a vertex. The test condition and the flow of f is indicated below each configuration.

Definition 3 *A dependency graph of cuts is a directed graph where each vertex is a cut. An arc exists between two vertices if the flow of the to-vertex can be computed from the from-vertex. If a cut is implicit, there are 5 incoming arcs. Explicit cuts have no incoming arcs.*

Since the flow of each implicit cut can be computed by different sets of surrounding cuts, there can be many possible dependency graphs. We seek a form of dependency graph that allows us to compute the flows of all implicit cuts. A *loop-free* dependency graph accomplishes this purpose (Fig. 7).

One way to construct a loop-free dependency graph is as follows. Let $\sigma(c)$ be the number of explicit cuts the cut c intersects. Note that $\sigma(c) = 0$ if c is an explicit cut. If for each arc (u, v) in a dependency graph G, we can guarantee $\sigma(u) < \sigma(v)$, then G is loop-free. For this to work we also have to show that 1) for any implicit cut, there exists at least one set of 5 cuts from which its flow can be calculated, and 2) all these 5 cuts has a lower σ. Once we obtained the dependency graph, we can compute the flow of any implicit cut from the flows of all explicit cuts. This allows us to do, for example, incremental recalculation of the flows of all affected implicit cuts whenever an explicit cut's flow changes.

Let the *corridor* be the sequence of triangles an implicit cut intersects (This is part of the cut-based encoding). A corridor is bounded by a left and a right *wall*, which are two paths not intersecting the cut but ending at the same vertices as the cut. For a cut from vertex u to vertex v, let the vertices on the left wall be $(u, l_1, l_2, \ldots, l_m, v)$ and on the right wall be $(u, r_1, r_2, \ldots, r_n, v)$. We seek a pair (l_i, r_j) such that cuts $\overline{l_i r_j}$, $\overline{u l_i}$, $\overline{u r_j}$, $\overline{l_i v}$, $\overline{r_j v}$ all exist and their σ number are all less than $\sigma(\overline{uv})$.

We can simply check all mn combinations for the five cuts. Since we only need to compute the dependency graph once, this step is not critical. More importantly, we need to show that 1) there exists a pair of vertices on each wall with all the 5 cuts, and 2) all the 5 cuts have lower σ than the given cut.

Consider a vertex l on the left wall of a cut c from vertex u to v which is the closest to the cut line (Fig. 12). The distance between a vertex and a cut line is the distance between its corresponding terminal and the cut line or between the nearest point on the closure of the corresponding partial boundary and the cut line. It can be shown that l is visible from u and v. Assume not, then suppose some vertex l' blocks l from u. Consider the triangle luv. Its height is the distance between l and $\overline{uv}$, which is the maximum among points on lu and lv. Hence l' must be strictly closer to $\overline{uv}$ in order to block l from u. This is contradict to the proposition that l is the closest vertex to $\overline{uv}$.

Since l is visible from u and v, there is a cut between l and u and between l and v. By the same argument, there are cuts $\overline{ru}$ and $\overline{rv}$ where r is the closest vertex on the right wall.

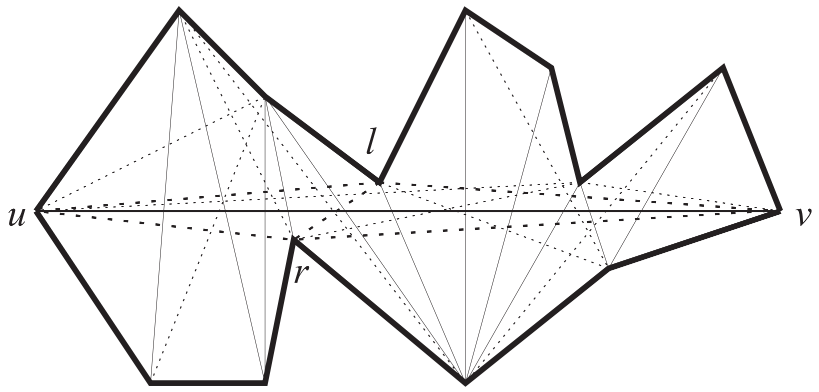

Figure 12: Corridors, walls, closest vertices and dependent cuts

Finally, there is a cut $\overline{rl}$ because there is no vertex within the quadrilateral *urvl* so r and l must be visible to each other. We call the cut $\overline{rl}$ the *diagonal* cut and $\overline{ru}$, $\overline{rv}$, $\overline{lu}$ and $\overline{lv}$ *enclosing* cuts. We have the following lemma.

Lemma 8 *Given any implicit cut $\overline{uv}$, there exists a set of five cuts $\overline{ul}$, $\overline{lv}$, $\overline{ur}$, $\overline{rv}$ and $\overline{rv}$ where l is the closest vertex from the cut $\overline{uv}$ on its right wall and r is the closest vertex on its left wall.*

Now we show that all these five cuts intersects fewer explicit cuts than the given implicit cut. If the diagonal cut is an explicit cut, all the enclosing cuts intersects at least one explicit cut less than $\overline{uv}$ because $\overline{uv}$ intersects the diagonal cut but the enclosing cuts do not.

The argument is more technical when the diagonal cut is implicit. We refer interested readers to Appendix B for details. We proved the following lemma.

Lemma 9 *Given any implicit cut c, its diagonal cut and all its enclosing cuts intersects fewer explicit cuts than c.*

The following theorem summaries the results of this section.

Theorem 10 *The flow of all implicit cuts can be computed by building and traversing a loop-free dependency graph (Def. 3).*

Fig. 13 is a simple breadth-first search algorithm to compute all the flows of affected implicit cuts when the flow of some explicit cuts changed.

Since a cut-based encoding records the sequence of explicit cuts each implicit cut intersects, the σ of an implicit cut takes constant time to compute (assuming that the length of the sequence is stored). Thus sorting all implicit cuts by their σ requires $O(m \log m)$

Algorithm 2 (Inc-DRC)

```
Algorithm Inc-DRC (Dependency graph $G$, Set of changed explicit cuts $S$)
    Add $S$ to queue $Q$
    while $Q$ not empty do
        $e \leftarrow$ dequeue($Q$)
        Update the flow of $e$ using Lemma 8 and 9
        $Q \leftarrow$ enqueue($Q$, children of $e$)
    endfor
```

Figure 13: Algorithm Inc-DRC

time where m is the total number of implicit cuts. Building the dependency graph requires finding the diagonal cut and the enclosing cuts for each implicit cut. If most implicit cuts are short (Appendix A), it is more efficient to check all pairs of vertices between the walls than computing the closest vertices to the cut on each wall. Hence it takes $O(mk^2)$ time where k is the number of vertices of the longest wall. Traversing the graph takes only $O(m)$ time because each node in the graph is only visited once and it takes constant time to decide the topology within a triangle (Triangle Encoding Theorem). The size of the graph depends on the number of implicit cuts, which depends on the total number of cuts because the number of explicit cuts is fixed with respect to the number of obstacles. Appendix A gives an estimation on the total number of critical cuts.

The power of our design rule check algorithm lies in the last step. As long as the dependency graph is unchanged, we can update the flow of all the depending implicit cuts whenever there is any change in the flow of any explicit cut. This algorithm can be expanded easily to wires with variable width and spacing so that the sequence of wires intersecting any cut, implicit or explicit, can be computed efficiently.

8 Conclusion

This paper presented advances in three important areas in general area routing. For pins arranged in a grid, the rectilinear wiring metric has asymptotically less number of critical cuts than octilinear and Euclidean metrics. Even under octilinear and Euclidean metrics, the expected number of cuts to check is a lot less than the worst case. Shadowing and a finite wire pitch contributed to reduce the number of critical cuts. The latter also tends to eliminate long critical cuts, which costs more to check than short cuts.

A second contribution of this paper is a *practical* wiring model based on cuts. This is the first wiring model that considers obstacles of arbitrary size and shapes. Pins are also allowed

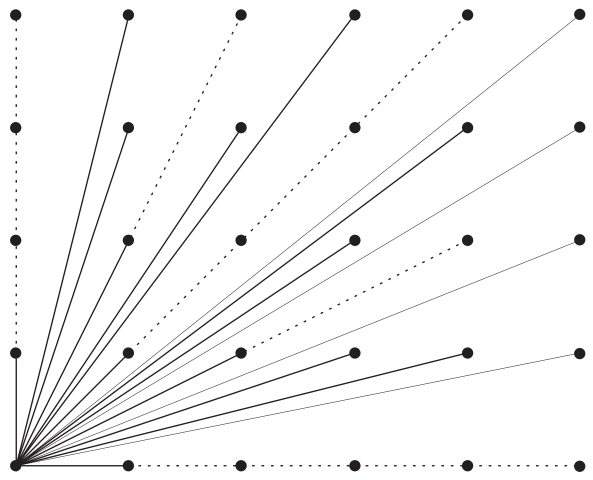

Figure 14: All the points visible from the origin on a 6×5 grid.

on the boundary of obstacles. This model can be directly applied to standard cell, MCM or PCB routing. This model is particularly useful for performance-driven routing because it has no restrictions on wire width and spacing, pin and obstacle placement, shape of obstacles, and wiring metric. By recording only the topological routing instead of the exact geometry of the wires, the encoding is compact and clean. Coupled with the dependency graph, another contribution of this paper, the encoding allows fast computation of the flow of any cut and enhance can decide the routability of a topological routing. The flow computation is also incremental in nature so the encoding is highly suitable for iterative improvement schemes such as simulated annealing. Our approach pushed all the work into generating a TEG, which depends only on the relative positions of the obstacles and terminals. Clearly our philosophy here is that the components are not nearly moved as much as ripping-up and rerouting wires. Comparing to other wire-based schemes, this is a sure win.

A A Realistic Estimation of the Number of Critical Cuts

For m features, in the worst case we need to check $m(m-1)/2$ cuts. This is the case when all features are arranged in a circle. In a realistic routing problem, features are more likely to be pads arranged in grids. In the following we show that the number of features visible from a feature in a grid is linear to the number of grid points. The number of cuts to check is further reduced due to the finite size of wire width and spacing. Finally, the number of cuts is reduced to eight per grid point if we use rectilinear wiring metric.

Suppose all features are points on a regular grid with unit pitch. The grid is n features wide. Consider the number of points visible from the point at the lower-left corner, which is at the origin. We can conveniently address the grid by integer coordinates. Recall that a point is visible from another point if the straight line segment between these two points does not intersect or touch any other points. A point at (h, k), $h > 0$, $k > 0$ hides all the points that lie on the same straight line from $(0, 0)$ to (h, k). These points have the coordinates (lh, lk) where l is an integer greater than 0 (Fig. 14). If we associate each point (h, k) with the fraction k/h, the number of points visible from the origin, ν, is then the number of irreducible fractions with numerators and denominators less than or equal to n. By symmetry, $\nu = 2\Phi(n)$ where $\Phi(n)$ is the number of irreducible fractions in $[0 \ldots 1]$ with denominator less than or equal to n. It can be shown that[6, pp. 139] $\Phi(x) = 3x^2/\pi^2 + O(x \log x)$. Hence $\nu \in \Theta(n^2)$, i.e., linear to the number of points. This is the same order of complexity as in the worst case but the average number of visible points decreased from $n - 1$ to $6n/\pi^2 \approx 0.608n$.

A further refinement of this model considers the effect of the finite wire width and spacing. Consider the grid point $(5, 2)$ in Fig. 15. If the minimum wire pitch (width plus spacing) is 0.1, then the capacity of the cut $(0, 0) - (5, 2)$ is $\lfloor\sqrt{25 + 4}/0.1\rfloor = 53$. The total capacity of the cuts $(0, 0) - (2, 1)$ and $(2, 1) - (5, 2)$ is $\lfloor\sqrt{4 + 1}/0.1\rfloor + \lfloor\sqrt{9 + 1}/0.1\rfloor = 22 + 31 = 53$. Therefore if the latter two cuts are safe, the former is safe. Hence we can exclude the former cut from our check list. Intuitively, the larger the minimum wire pitch, p, the more long cuts we can throw away. In the following we derive a relation between the number of cuts to be checked from the origin against p.

Any irreducible fraction can be expressed as $(m + m')/(n + n')$ where m, m', n and n' are positive integers and $m'n - mn' = 1$ [6, pp. 118]. The three fractions, $m/n < (m + m')/(n + n') < m'/n'$ are *consecutive* fractions in the sense that there are no other fractions of denominator less than or equal to $n + n'$ between the gaps. We will show that the sum of the lengths of the vectors $(n\ m)^T$ and $(n'\ m')^T$ are the closest to the length of the vector $(n + n'\ m + m')^T$.

Lemma 11 *Given two distinct fractions m/n and m'/n' with $m/n < m'/n'$, the angle ϕ at (n', m') subtended by the line segment $(0, 0) - (n + n', m + m')$ is the largest if $m/n, (m + m')/(n + n'), m'/n'$ are consecutive.*

PROOF: Refer to Fig. 15. If θ and θ' are the angles of elevation of the vectors $(n\ m)^T$ and $(n'\ m')^T$ respectively, then $\tan\theta = m/n$ and $\tan\theta' = m'/n'$. Since $\phi = \pi - (\theta - \theta')$, it is maximized when $\theta - \theta'$ is minimized. Indeed, since $m/n, (m + m')/(n + n'), m'/n'$ are consecutive, $\tan\theta' - \tan\theta = m'/n' - m/n$ is minimal. Therefore $\theta' - \theta$ is minimal because $0 \leq \theta', \theta \leq \pi/2$. □

By Lemma 11, we only need to consider the triangle $(0, 0)$, (n, m), $(n + n', m + m')$ since

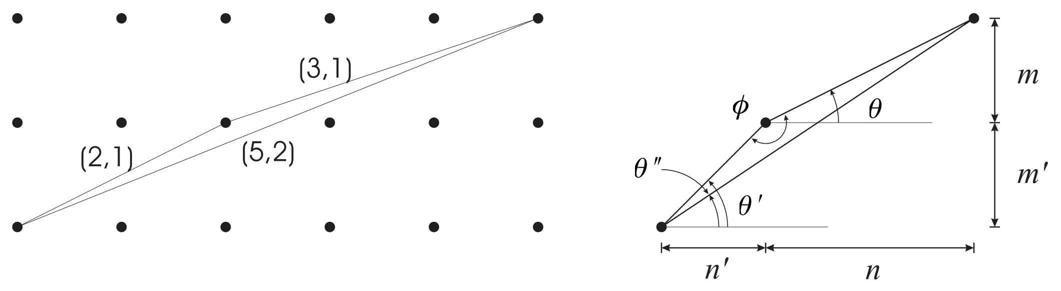

Figure 15: The cut $(5,2)$ has the same flow as the sum of cuts $(3,1)$ and $(2,1)$. If the wire pitch is large enough, the capacities will be the same too because flow has to be integer. Therefore it is sufficient to check $(3,1)$ and $(2,1)$.

any other triangle will be less "skinny" than this one. We consider z, the difference in the sum of the length of the two vectors $(n\ m)^T$ and $(n'\ m')^T$ and the vector $(n+n'\ m+m')^T$.

$$z = \sqrt{m^2+n^2} + \sqrt{m'^2+n'^2} - \sqrt{(m+m')^2+(n+n')^2}.$$

If z is less than one wire pitch, we can eliminate the cut $(0,0)-(n+n', m+m')$ from the check list. In the following we develop an asymptotic estimation on z as m, n, m' and n' becomes very large.

We start with z^2.

$$\begin{aligned} z^2 &= 2\Big\{m^2+n^2+m'^2+n'^2+mm'+nn'+\sqrt{(mm'+nn')^2+1} \\ &\quad - \sqrt{(m^2+mm'+nn'+n^2)^2+1} - \sqrt{(m'^2+mm'+nn'+n'^2)^2+1}\Big\}, \end{aligned}$$

using the relation $(m'n-mn')^2 = m'^2n^2+m^2n'^2-2mm'nn' = 1$. Since the coordinates are large, we use $\sqrt{x^2+1} = x(1+1/(2x^2)+O(1/x^4))$. Then, after simplification,

$$\begin{aligned} z^2 &\approx \frac{1}{mm'+nn'} - \frac{1}{m^2+mm'+nn'+n^2} - \frac{1}{m'^2+mm'+nn'+n'^2} \\ &= \frac{1}{(mm'+nn')(m^2+mm'+nn'+n^2)(m'^2+mm'+nn'+n'^2)}. \end{aligned}$$

Let $r^2 = m^2+n^2$ and $r'^2 = m'^2+n'^2$. Since $(mm'+nn')^2+1 = r^2r'^2$,

$$\begin{aligned} z^2 &\approx \frac{1}{\sqrt{r^2r'^2-1}(r^2+\sqrt{r^2r'^2-1})(r'^2+\sqrt{r^2r'^2-1})} \\ &\leq \frac{1}{r^2r'^2(r+r')^2}. \end{aligned}$$

Hence,

$$z \approx \frac{1}{rr'(r+r')}.$$

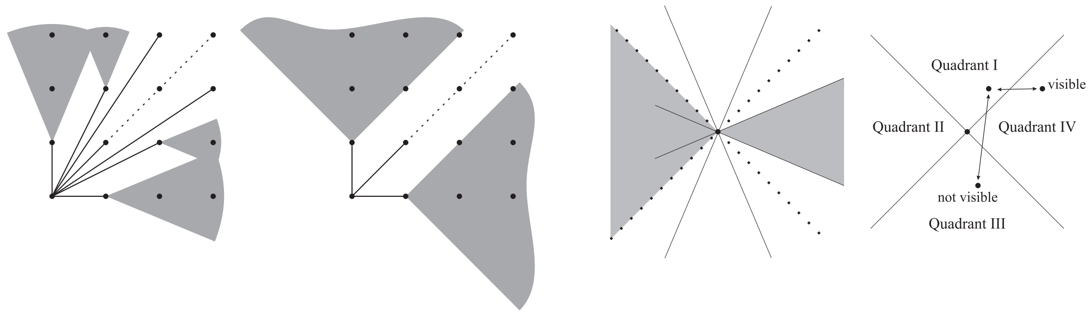

Figure 16: $2(n-1)+1$ points are visible under shadowing of octilinear metric. Only 3 points are visible under rectilinear metric. The two right diagrams shows the regions of visible points. For octilinear metric, all points in the shaded area at the right is shadowed by the point at the center to the points in the shaded area bounded by dotted lines at the left. The other three directions are similar. The rightmost diagram shows shadowing for rectilinear metric. Two points within neighboring quadrants are visible. Points in opposite quadrants are not visible.

We do not have to check cuts longer than such that z is less than or equal to the wire pitch p. Therefore $\max(r, r')$ defines a radius within which we need to check all cuts. In Euclidean metric the number of cuts we need to check for an n-size grid is $O(n^2)$. Since z is in $O(1/r^3)$, we have the following lemma.

Lemma 12 *The number of cuts we need to check at the origin for wire pitch p is $O(1/p^{3/2})$.*

If we are willing to sacrifice some routability and use a piecewise linear wiring metric, we can potentially reduce the number of cuts to check to a constant at each point. A point q is *shadowed*[7][3, pp. 46] by another point p if $\operatorname{cap}\overline{q0} = \operatorname{cap}\overline{r0} + \operatorname{cap}\overline{qr}$, where 0 is the origin. Figure 16 shows shadowing for octilinear and rectilinear wiring metric. We define the *shadowed visibility graph* to be the set of cuts we need to check under shadowing. A pair of points are *4-visible* if they are visible under rectilinear shadowing. Similarly, we define *8-visible* for the octilinear metric and *∞-visible* for Euclidean metric. We simply use *visible* when it is applicable for all routing metrics. It is straightforward to show that 4-visibility implies 8-visibility, which implies ∞-visibility.

On a grid, for octilinear metric, the points $(1,0)$, $(0,1)$, $(1,1)$, $(i+1,i)$ and $(i,i+1)$ for $i > 0$ are 8-visible from the origin. For a grid of size $n \times n$, the number of 8-visible grid points is $2(n-1)+1 \in O(n)$. Applying the finite pitch argument the cut size further reduces to $O(1/p^3)$. For rectilinear metric, only the three nearest neighbors are 4-visible, i.e., $O(1)$. In general, the critical cut set consists of cuts to the nearest eight grid points for each grid

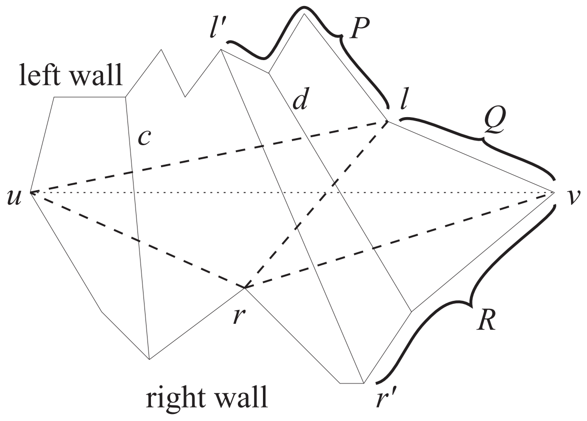

Figure 17: Proof of Lemma 9

point. This is an asymptotic reduction in the number of cuts to check. It explains the ease of design rule check in early routing systems. These systems represented all obstacles as grid points and used the rectilinear wiring metric. For octilinear wiring metric, the size of the critical cut set is asymptotically the same as Euclidean wiring metric. This is the fundamental reason for the difficulties in extending grid-based, rectilinear routing systems to true octilinear or all-angle systems.

B Proof of Lemma 9

Suppose the implicit cut under consideration is from vertex u to vertex v. The vertices l and r are the closest vertices on the left and right walls, respectively, to the cut line. If the diagonal cut $\overline{lr}$ is implicit, then there is at least one intersecting explicit cut since a TEG is maximally planar. Call this cut $\overline{l'r'}$. Without lost of generality, let l' be in the path from u to l and let r' be in the path from r to v.

First we observe that if an enclosing cut or the diagonal cut intersects some explicit cut c, the dependent cut $\overline{uv}$ also intersects the same explicit cut c. Otherwise l or r will not be on their respective wall of the dependent cut.

Now we consider the number of explicit cut intersections of $\overline{lv}$. If it is explicit, then $\sigma(\overline{lv}) = 0$ and is less than $\sigma(\overline{uv})$. Otherwise, the cut $\overline{uv}$ intersects any explicit cut $\overline{lv}$ intersects plus at least the explicit cut $\overline{l'r'}$ (Fig. 17). Hence $\sigma(\overline{lv}) < \sigma(\overline{uv})$. The same argument applies to $\overline{ru}$.

All we need to show now is that $\overline{lu}$, $\overline{rv}$ and $\overline{lr}$ intersects fewer explicit cuts than $\overline{uv}$. For $\overline{lu}$ (argument for $\overline{rv}$ is similar), we consider the loop bounded by l', l, v and r'. Since a TEG is maximally planar, there is at least one explicit cut within this area. We consider the paths P from l' to l^-, Q from l to v^- and R from v^+ to r'. The vertex l^- is the vertex before l on the left wall from u to v. Similarly, v^- is the vertex before v on the left wall. v^+ is the vertex before v on the right wall.

An explicit cut between a pair of vertices in P and R is intersected by both $\overline{ul}$ and $\overline{uv}$ so it contribute to both σ numbers. An explicit cut between a pair of vertices in P and Q is not possible because P and Q are part of the left wall. The only remaining combination is from Q to R. An explicit cut from Q to R intersects only $\overline{uv}$ but not $\overline{ul}$ so if one explicit cut exists then $\sigma(\overline{uv}) > \sigma(\overline{ul})$.

If Q has more than one vertex or if $\overline{lv}$ is implicit, then there is at least one explicit cut from Q to R that intersects $\overline{uv}$ because any explicit cut from Q must end on some vertex in R. On the other hand, if Q is just the vertex l and $\overline{lv}$ is explicit, there is an explicit cut from l to some vertex in R too, due to the maximal planar property of the TEG and the fact that l is on a wall.

Therefore in all cases $\sigma(\overline{uv}) > \sigma(\overline{ul})$. Using the same argument we can prove that $\sigma(\overline{uv}) > \sigma(\overline{rv})$ too. For $\overline{rl}$, we note that every explicit it intersects also intersects the cut $\overline{uv}$ so $\sigma(\overline{rl}) \leq \sigma(\overline{uv})$. The argument about explicit cuts between Q and R in the previous paragraph applies here too. Therefore $\sigma(\overline{rl}) < \sigma(\overline{uv})$. Thus we complete the proof of Lemma 9.

References

[1] D. Staepelaere, J. Jue, T. Dayan, and W. W.-M. Dai, "Surf: A rubber-band routing system for multichip modules," *IEEE Design and Test of Computers*, December 1993.

[2] J. Valainis, S. Kaptanoglu, E. Liu, and R. Suaya, "Two-dimensional IC layout compaction based on topological design rule checking," *IEEE Trans. Computer-aided Design*, vol. 9, pp. 260–275, March 1990.

[3] F. M. Maley, *Single-layer wire routing and compaction.* Cambridge, MA: MIT Press, 1990.

[4] K.-Y. Khoo and J. Cong, "An efficient multilayer MCM router based on four-via routing," *IEEE Trans. Computer-aided Design*, vol. 14, pp. 1277–1290, October 1995.

[5] J. D. Cho, K.-F. Liao, S. Rajie, and M. Sarrafzadeh, "M^2R: Multilayer routing algorithm for high-performance MCMs," *IEEE Trans. Circuits and Systems I*, vol. 41, pp. 253–265, April 1994.

[6] R. L. Grahm, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A foundation for Computer Science.* Reading, MA: Addison-Wesley, 2nd ed., 1994.

[7] R. Cole and A. Siegel, "River routing every which way, but loose," in *Proc. 25th Ann. Symp. Foundation of Comp. Sci.*, (Singer Island, FL), pp. 65–73, IEEE, IEEE Computer Society Press, October 1984.

[8] R. P. Bazylevych, E. Zamora, and N. F. Storozenko, “The flexible routing algorithm for PCB,” *Visnyk Lvivskoho Politekhnichnoho Instytutu*, vol. N76, pp. 83–88, 1973. In Ukrainian.

[9] R. P. Bazylevych, E. F. Zamora, N. F. Storozenko, and R. Pelke, “Flexible Literzugverlegung fur Zweiseitige Leiterplatten mit Hilfe der EDVA ’m-222’,” in *XX Internat. Vissenschaftliches Kolloquium*, (Technische Hochschule Ilmenau), pp. 159–162, 1975. In German.

[10] R. P. Bazylevych and R. Pelke, “Probleme der Optimierung der Leiter zugver legung bein Rechnergestiitzten Leiterplattenentwurf,” in *23 Intern. Wiss. Koll.*, (TH Ilmenau), pp. 117–120, 1978. In German.

[11] M.-F. Yu, J. Darnauer, and W. W.-M. Dai, “Interchangeable pin routing with application to package layout,” in *Proc. Intl. Conf. Computer-aided Design*, (Santa Clara, CA), IEEE, November 1996.

[12] S. K. Ghosh and D. M. Mount, “An output sensitive algorithm for computing visibility grpahs,” *SIAM J. of Computing*, vol. 20, pp. 888–910, October 1991.

[13] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction.* Texts and monographs in computer science, New York, NY: Springer-Verlag, 1985.