Planar Interchangeable 2-Terminal Routing

Man-Fai Yu
Joel Darnauer
Wayne Wei-Ming Dai

UCSC-CRL-95-49
October 19, 1995

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

ABSTRACT

Many practical routing problems such as BGA, PGA and test fixture routing involve routing two-terminal nets on a plane with exchangeable pins. In this paper, we unify these different problems as instances of the Planar Interchangeable 2-Terminal Routing (PI2TR) problem. We formulate the problem as flows in a routing network. Secondly, we show that PI2TR is NP-complete by reduction from satisfiability. Finally, we show experimental evidence that a simple min-cost flow heuristic considering only the most important cuts in the design can quickly produce routable results in most practical cases. The flow formulation can be generalized to multiple layers with no additional vias.

Keywords: package routing, planar routing, routability, network flow, exchangeable pins

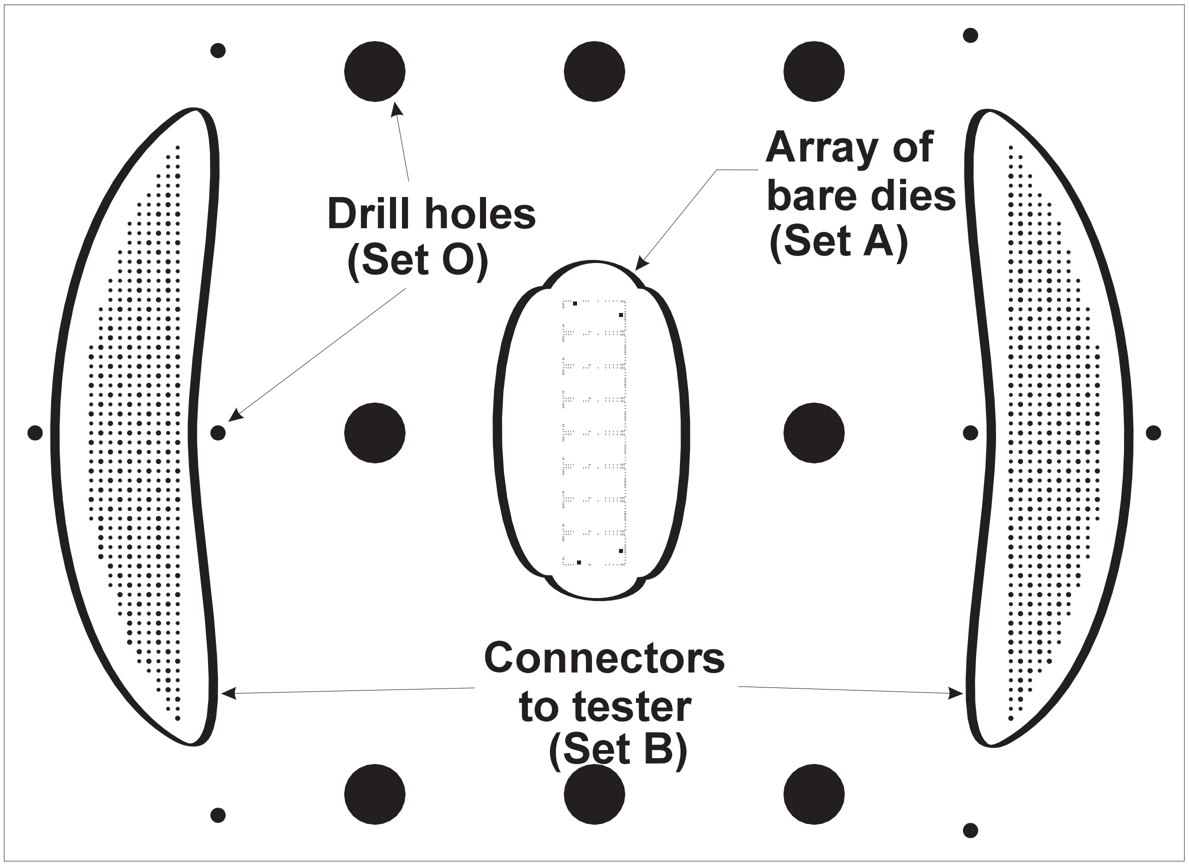

Figure 1: An unrouted example of routing with exchangeable pins. Each pad in the center must be connected to one pad on the edge connector.

1 Introduction

In this paper we explore the problem of routing with the freedom of exchangeable pins. For example, in Ball Grid Array (BGA) or Pin Grid Array (PGA) package design, we often wish to connect each chip pad to a single package pin, but we may not always care which pad is connected to which pin. This same type of problem also occurs in the design of test fixtures, footprint escape patterns, and to some extent in routing signals inside an ASIC to one of the available pads on the periphery. Figure 1 shows a typical problem instance before routing. We want to connect all pads in set A to set B while avoiding all obstacles. We do not care which pin in B a pad is connected as long as there is *some* pin.

Conventional routers cannot address these problems because they require the user to generate a complete netlist before the routing process can be started. In addition to adding an extra step to the design process, the choice of pin assignment can have a big impact on quality of the routing. Often a pin assignment is chosen that cannot be realized. Rather than performing pin assignment and routing as two separate steps, what is needed is a router that performs these two steps simultaneously.

Specific package routers like PGA or BGA routers[1, 2, 3, 4] have been developed that take advantage of special geometries of their respective problems and the exchangeable freedom of the pins. Although these methods are exceptionally efficient, they have a number of limitations:

- They rely on the symmetry of the arrays and rings to generate solutions and cannot cope with missing, skewed, or off-grid pads.
- Even in these very symmetric cases, these package routers[3, 2] may still produce designs that are $\sqrt{2}$ times more dense than the best possible designs. In particular, the algorithms cannot always tell if they have generated a routable solution, and have no strategy for changing an unroutable solution into a routable solution.
- They do not consider the case where the number of pads in the two sets of pins to connect is unequal.
- They do not take into account the presence of obstacles.

In this paper, we offer the following practical contributions. First, the package routing problems discussed above have a natural formal description, which we call *Planar Interchangeable Two-Terminal Routing* (PI2TR). We show that although we can take advantage of the freedom of exchangeable pins, PI2TR is NP-complete. Finally, we show that despite the NP-completeness, a min-cost flow heuristic can produce good results by considering only some cuts in a design.

1.1 Formal Problem Definition

The fundamental features of the package problem routing are two sets of pins placed arbitrarily in a single plane that we wish to connect to one another. We also need to model the routing area and any obstacles that are present, as well as the particular design rules permitted by the wiring. Accordingly, we define an instance of the Planar Interchangeable 2-Terminal Routing (PI2TR) problem as follows:

Definition 1: Planar Interchangeable 2-Terminal Routing

An instance of the Planar Interchangeable 2-Terminal Routing (PI2TR) problem is a 6-tuple of (b, A, B, O, w, s) *where:*

b *is a polygon representing the bounds of the routing area.*

A *is a set of polygons for one class of pins.*

B *is a set of polygons for the other class of pins.*

O *is a set of polygons representing obstacles.*

w *is a positive integer for the minimum wire width.*

s *is a positive integer for the minimum wire spacing.*

A, B, O are non-overlapping polygons inside b. Without loss of generality $|A| \leq |B|$.

A solution to the problem is a *detailed routing* of the design, i.e a set of wire paths that connects each pad in A to a unique pad in B and obey the width and spacing rules. [1] PI2TR is in NP because we can check a given detailed wiring in polynomial time by verifying that each A is connected to a B, and that the design rules are correct. The problem is now to define the set of detailed routings that are potential solutions, and then to choose one that is feasible.

Maley[5] showed that the routability of any detailed routing can be checked by checking the feasibility of its corresponding *topological routing.* A topological routing is the equivalence class of detailed routing under homotopic transformation. Two detailed routings that can be homotopically transformed from one to the other belongs to the same equivalence class and thus has the same topological routing.

The topological routing is routable if the total number of wires flowing through cuts between any pair of features (the *flow of the cut*) is less than the maximum number of wires that could be accommodated in the best case (*the capacity*).

Because there are efficient algorithms for finding a correct detailed routing from a topological routing[6], we will consider a feasible topological routing a solution for PI2TR.

2 Network flow formulation of PI2TR

[1] For simplicity, we will assume for now that the wires are permitted to run at any angle, although the formulation could include another parameter to characterize the wiring style.

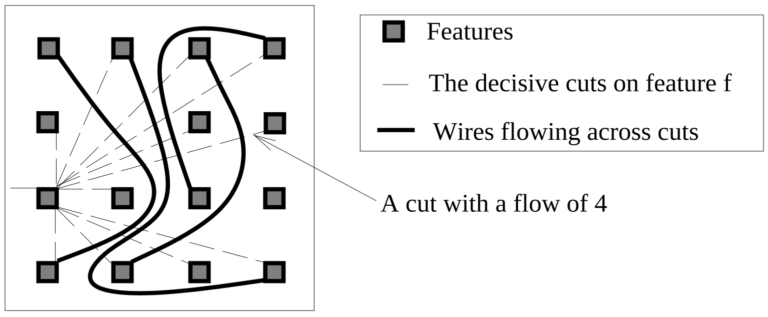

Figure 2: A topological routing with some cuts and flows illustrated

In this section we develop a network flow formulation for PI2TR. We showed that any flow assignment can be transformed to a topological routing.

2.1 The Routing Network

In the case of 2-terminal nets, we may recognize some similarities between a topological routing and a network flow. For example, a net can be modeled as a flow from a source (a terminal) to a sink (the other terminal). Flow is conserved at nodes that are not source nor sink. Nets are also conserved because they only terminate at pins. Each source originate one unit of flow and each source terminate one unit of flow. Similarly, each pin either originate or terminate a net. To make these ideas more concrete, we consider the *routing network* of a design.

Definition 2: *If Δ is the dual of some triangulation of the vertices of $A \cup B \cup O$ and the boundary b. A* Routing Network *$T(V, E, s, t)$ is a network where $V = \Delta \cup A \cup B$. E contains a pair of opposite arcs between each pair of adjacent triangles and one pair of opposite arcs between each vertex in $A \cup B$ and the triangles they are incident to. Each arc has a non-negative cost. The arcs originating from a pad of set A have unit capacities and arcs to a pad of set A have zero capacities. The opposite is true for set B. Finally, the arcs connecting triangles have a capacity equal to the capacity of the cut between the two vertices in the triangulation dual in both directions.*

Note that the vertices in Δ are triangles. The reader should note that there are many possible routing networks for a given problem instance. Figure 3 shows a routing network of a PGA package. Each line segment in the figure represents a pair of opposite arcs in the network. For each net one of its terminals is adjacent to the supersource and the other to the supersink. The flow on the

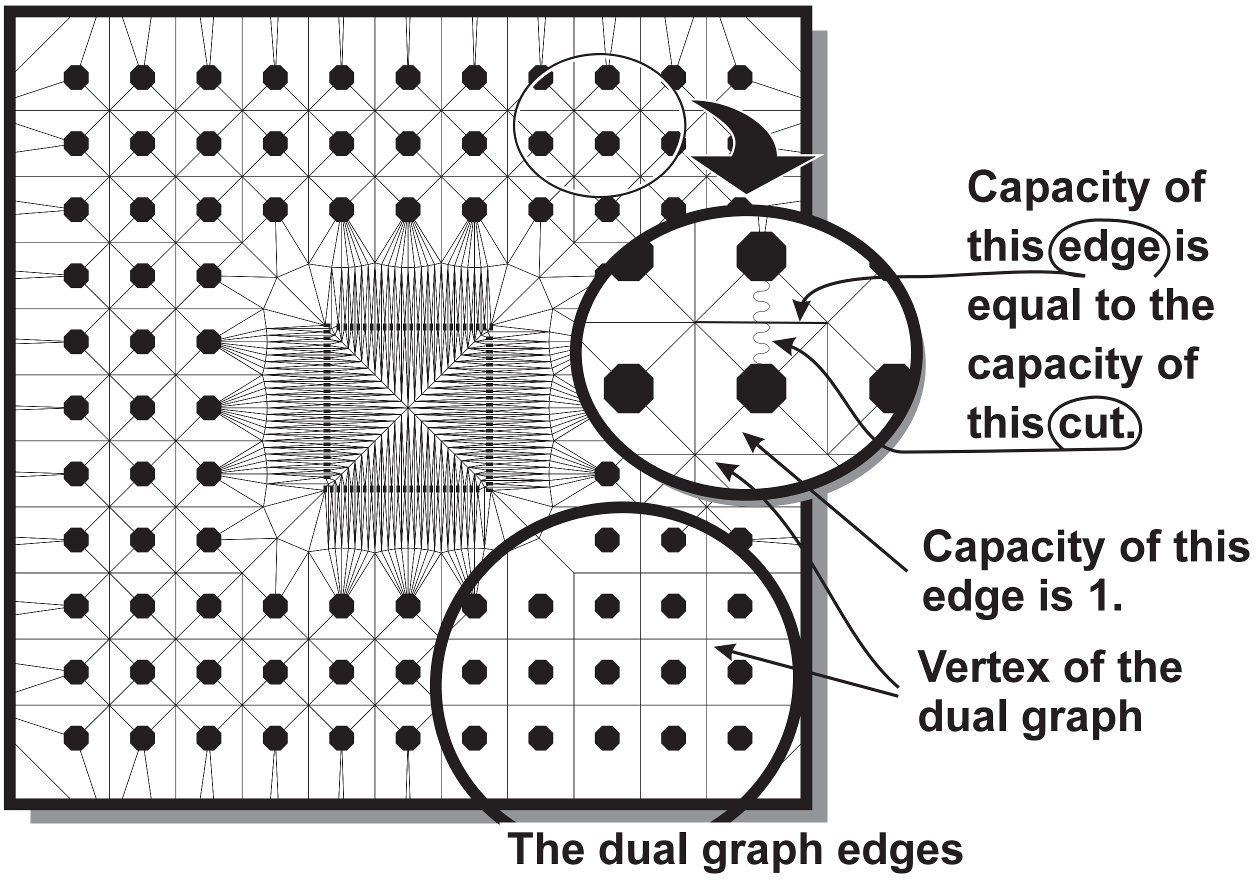
Capacity of this edge is equal to the capacity of this cut.
Capacity of this edge is 1.
Vertex of the dual graph
The dual graph edges

Figure 3: The Routing Network of a PGA package

arcs of the routing network is the number of nets that intersect the corresponding cut. Any flow assignment on the routing network are conservative.

- The flow entering a vertex in A must be -1.
- The flow entering a vertex in B must be either 0 or $+1$. (Some vertices in B may not be used.)
- The net flow entering a vertex in Δ must be 0. (Wires end at pads, not in triangles.)

We can show that flow assignment on the routing network of a design can be transformed into a topological routing of the design.

Theorem 1: *A flow assignment on a routing network of a design can be transformed into a topological routing of the same design.*

Proof: We can transform the flow one triangle at a time. In each triangle, due to the conservation condition, we can always find a proper topological routing in the triangle. We delay the discussion of specific cases to Section 4 where a specific algorithm is developed. The overall topological routing can be formed by patching all the triangles together. □

2.2 Routability of the topological routing

Maley[5] showed that only a certain set of cuts, the *decisive cutset*, needs to be checked for a topological routing to determine its routability. The basic idea is that the safety of the shortest cut between two features implies the safety of all the cuts between them. If this cut is safe, then all cuts between these two features are safe, and vice versa. If there are N features, then we need to check at most $N(N-1)/2$ cuts. Thus, the number of cuts in the decisive cut set is bounded by $O(|A+B+O|^2)$.

In the routing network only *some* cuts are represented as constraints. These cuts are the edges of the triangulation. There are only $O(N)$ cuts in a triangulation of N points while there are $O(N^2)$ cuts. It is obvious that many cuts are not explicitly represented by the triangulation. We call these *implicit cuts*. A flow solution only guarantees that all explicit cuts are safe but says nothing about implicit cuts. In the next section we will show that PI2TR is in fact NP-complete so no polynomial time algorithm that finds a routable topological routing is likely to be found.

3 PI2TR is NP-Complete

In this section we describe a proof of NP-completeness of PI2TR by reduction from satisfiability using the method of components[7].

PI2TR is in NP as was discussed in Section 1. It remains to show that PI2TR is complete for NP. Since 3-SAT is NP-complete, PI2TR is complete for NP if any instance of 3-SAT can be reduced to an instance of PI2TR.

Given an instance of satisfiability in the form of a boolean expression in clausal normal form, we can design a physical circuit layout in the form of a PLA with literals entering a switching matrix to be dispatched to a column of 3-input OR gates which feed into a large AND gate. Denote the

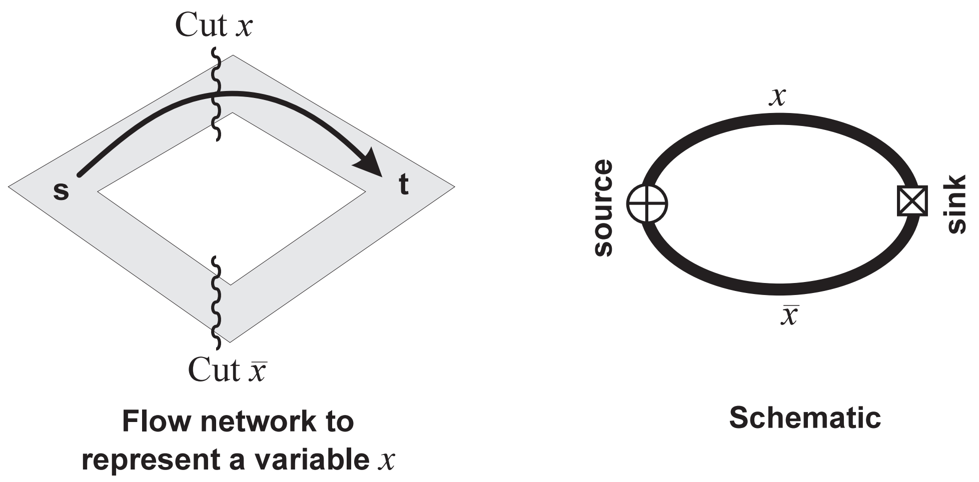

Figure 4: 0-1 variables can be represented by flow in selected channels

function computed by this circuit as F. This flow circuit has a number of components (gates, crosspoints) that is polynomial in the number of input terms. Our objective will be to show how we can implement this circuit as an instance of PI2TR and then use PI2TR to generate a satisfying assignment.

To achieve this we need a way of representing a boolean variable. In analogy to wires we will use a single long circular tube with sources and sinks alternating along its length. (Fig. 4). The tube may have some small openings, but none large enough to allow a whole wire to either flow in or out. All of the sources must connect to a neighboring sink inside the tube. All sources originate one unit of flow and all sinks terminate one unit. All tubes have the capacity of one. A feasible solution of PI2TR means we need integer flow assignment. The assignment of the variable x will correspond to the flow crossing a cut at a certain point in the tube. The complement of x is also available at some other location in the tube.

We will now construct a gate. The basic idea is to allow two tubes carrying flow to touch at some point along their length and to create a small opening, or window (Fig. 5). The window is narrow enough that flow cannot leak from tube to tube, but the presence of flow in one tube creates a bulge in the other tube that blocks it off. We can then create arbitrary constraints of the form $x + y < 1$. This basic switch element can be used to construct any logic gate. Fig. 6 shows a 2-input AND/NAND gate. Since any logic expression can be reduced to using NAND gate alone, we can realize any logic expression. Other gates can easily be constructed in a similar fashion.

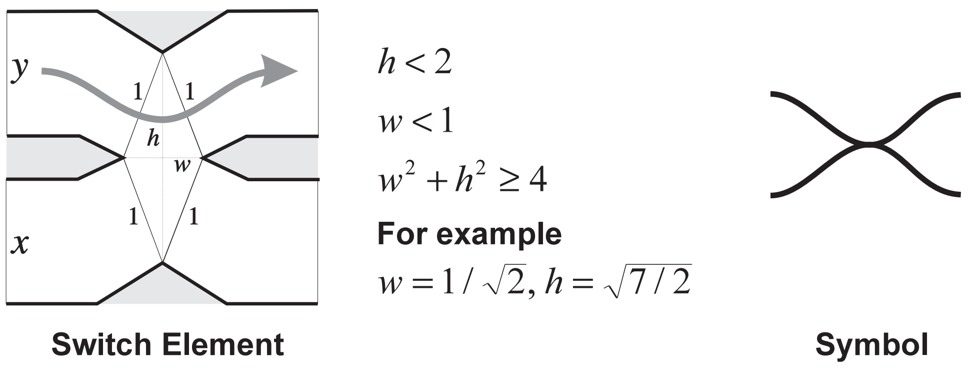

Figure 5: Intersecting channels allow constraints to propagate from one variable to another

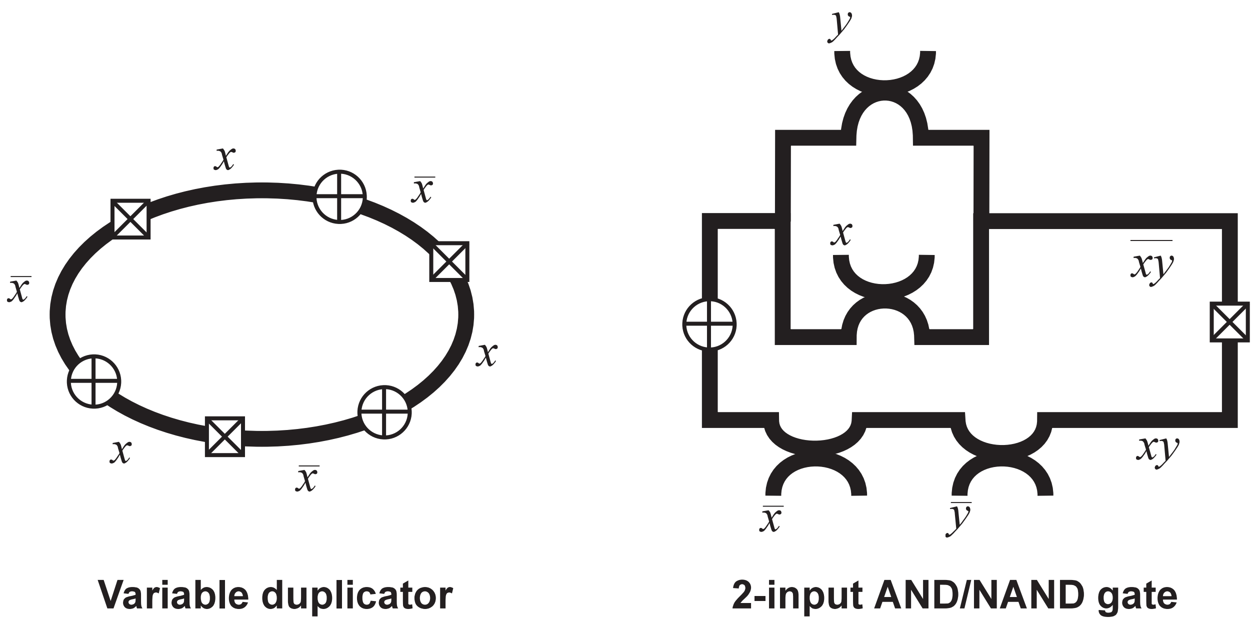

Figure 6: Intersections and flows can be used to construct logic gates

To be able to construct non-planar circuits, we need to cross one signal over another signal. Fig. 7 shows a special construct to swap two variables.

For a given assignment of flows, we can compute the resulting value of F. We need a way for the output of the final AND gate to affect whether or not the whole PI2TR instance has a feasible solution. Observe that our variable representation allows us to access both the positive and negated forms of a variable. That is, we can access either F or F' as we need. We can attach a tube with a single source, s, and a single sink, t. Call the flow in this tube z. Now we are free for z to share a window with the inverse of the function to be computed. If for some assignment of flow z is zero,

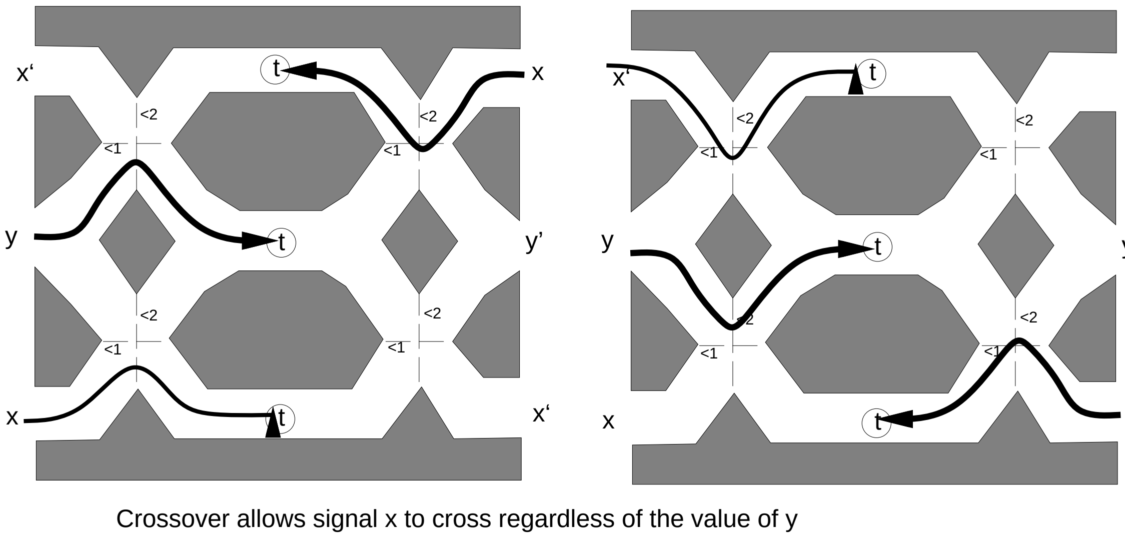
x'
x
y
y'
x
x'
t
<2
<1
Crossover allows signal x to cross regardless of the value of y

Figure 7: A special construction allows us to cross cycles over one another to construct non-planar circuits

then the source s has no connection to a sink and then the flow is infeasible. On the other hand, if for some assignment $z = 1$, then we know that $\bar{F} = 0$ or the flow z would be pinched off. Thus the only feasible flow is one where $F = 1$. We can then find the satisfying assignment by looking at the flow in the tubes corresponding to input variables. Thus PI2TR can be used to solve 3-SAT.

It is instructive to note that changing PI2TR in some ways has no impact on its NP-completeness. For example, allowing fractional-width wires allows solutions to circuits that would otherwise be infeasible. We could require that flow across cuts always flow in the same source-sink direction, but since it is always possible to arrange for the flow in a bottleneck to always travel in one direction, this does not help. The critical factor seems to be the fact that the two cuts in the bottleneck cross. If this can be eliminated, then the construction fails.

To summarize we have the following theorem.

Theorem 2: *PI2TR is NP complete.*

4 The Flow Router

In this section we describe in detail the **flow router** as a heuristic to solve PI2TR. We use the min-cost formulation. The router has three steps:

1. Building the routing network.
2. Solving the min-cost max-flow problem.
3. Transforming the solution into a topological routing.

4.1 Building the routing network

We used the incremental Delaunay triangulation described by Lu[8] to construct the triangulation, although any other algorithms are equally good. The Delaunay triangulation of $N = |A \cup B \cup O|$ points can be constructed in $O(N \log N)$ time[9]. The dual of the triangulation can be constructed in $O(N)$ time. Additional edges can be added in $O(|A \cup B|)$ time. The capacity of each edge is set as follows:

- If the edge is in the dual of the triangulation, then it represent a cut between two vertices. The capacity of the edge is

$$\left\lfloor \frac{\text{Length of cut} - \text{Pad sizes} - \text{Wire spacing}}{\text{Wire spacing} + \text{Wire width}} \right\rfloor.$$

  This is the number of wires that can intersect this cut without overflowing it.
- If the edge terminate at a pin, then the capacity is set to 1.
- The capacities of all edges of the supersink t and the supersource s are set to 1.

There is more flexibility in choosing the cost function. We choose the cost function to approximate the wire length of the final topological routing. Since the position of a wire intersecting a cut is equally likely along the cut, we choose the edge that represent the cut in the routing network to be the perpendicular bisector of the cut. The intersection of the three perpendicular bisectors of a triangle is the circumcenter of the circumcircle of the triangle. We therefore define the cost of an edge to be the distance between the circumcenters where the edge terminate. Other points in the triangle, such as the centroid, can be used too. Experiments show that the solutions between using the centroid and the circumcenter is not much different. This means that both are reasonably good estimators of real wire length.

4.2 The Min-cost Max-flow Algorithm

Algorithm 1 (BUILDUP):

```
Algorithm BUILDUP(Routing network $T$)
  for totalflow $\leftarrow$ 1 to $|A|$
    Path $p \leftarrow$ SHORTESTPATH($T$)
    if path is not found, return "$T$ is unroutable".
    Increment flow on all edges of $p$ by 1.
  endfor
```

Algorithm 2 (SHORTESTPATH):

```
Algorithm SHORTESTPATH(Routing network $T$)
  Set current cost on each vertex of $T$ to $\infty$.
  currcost($s$) $\leftarrow$ 0.
  Queue $q \leftarrow \{s\}$.
  Pass $n \leftarrow 0$.
  Vertex $z \leftarrow$supersink $t$.
  while nonempty($q$) and $n \leq$ total number of edges
    Vertex $v \leftarrow$ dequeue($q$).
    for $\forall w$ adjacent to $v$
      if flow($v, w$) < capacity($v, w$) and currcost($v$) + cost($v, w$) < currcost($w$)
        currcost($w$) $\leftarrow$ currcost($v$) + cost($v, w$), parent($w$) $\leftarrow v$.
        if $w$ is not in $q$, enqueue($q, w$).
      endif
      if flow($v, w$) < 0 and currcost($v$) − cost($v, w$) < currcost($w$)
        currcost($w$) $\leftarrow$ currcost($v$) − cost($v, w$), parent($w$) $\leftarrow v$.
        if $w$ is not in $q$, enqueue($q, w$).
    endfor
    if $v = z$, $n \leftarrow n + 1$, $z \leftarrow$ last element in $q$.
  endwhile
  if nonempty($q$) return "Path not found".
  find path by retracing back from the sink.
  return the shortest path.
```

Figure 8: Algorithm buildup

After the routing network is constructed, we run a min-cost max-flow algorithm on the network. The algorithm we used is based on the 'buildup' algorithm described in Papadimitrou and Steiglitz[10]. Informally, we try to find a minimum total cost assignment of flows for a given flow. In this case, the given flow is the maximum flow because the flow is equal to the number of connections, i.e. $|A|$. This flow is maximum because the sum of capacities of edges of the supersource is $|A|$. If we cannot push $|A|$ flow across the network, the design is unroutable. This is because a bottleneck of cuts in the triangulation is overflowed.

This algorithm requires a shortest path algorithm that handles negative-cost edges. We used the algorithm described in Tarjan[11]. This algorithm runs in $O(|V||E|)$ time. Note that the flow

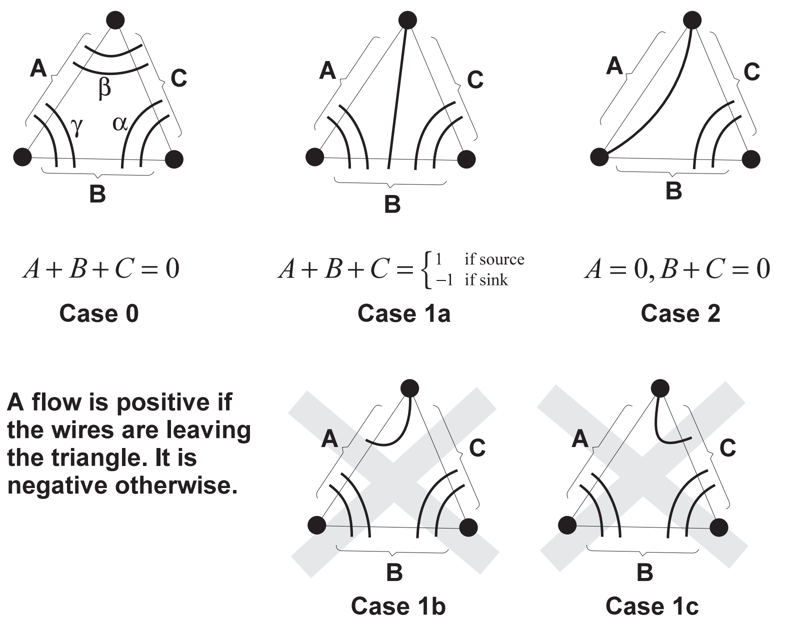
A
C
B
β
γ
α
$A+B+C=0$
Case 0
$A+B+C=\begin{cases}1 & \text{if source}\\ -1 & \text{if sink}\end{cases}$
Case 1a
$A=0, B+C=0$
Case 2
A flow is positive if the wires are leaving the triangle. It is negative otherwise.
Case 1b
Case 1c

Figure 9: Three cases of mapping flows in a triangle to a topological routing

on an edge can be positive or negative. The direction of the flow is always from the source to the sink. Fig. 8 shows both algorithms.

4.3 Transforming a flow solution to a topological routing

The last step in the flow router is to convert a min-cost flow solution to a topological routing. This can be done on a triangle-by-triangle basis. Fig. 9 shows the three possible cases of flow assignments on a triangle. Note that each edge of the triangle corresponds to a pair of opposite arcs in the routing network. In general the flow on both arcs are non-zero. We choose to simply the cases by cancelling out the flow on opposite arcs and only realize the net flow. Reducing the flow will certainly not violate any capacity constraint. We can only have three cases. Case 0 has no connection to any of the pins in the triangle. Case 1 has one connection and Case 2 has two. Since a topological routing is actually an equivalence class of homotopically equivalent detailed routings, Case 1b and Case 1c are redundant. Fig. 10 shows a homotopic transformation of a detailed routing involving a Case 1c triangle to a routing that does not use Case 1b or Case 1c triangles. Since the two routings are homotopically equivalent, they are the same topological routing.

In a case 0 triangle, we can compute the subflows α, β, γ from the flows A, B, C by the following

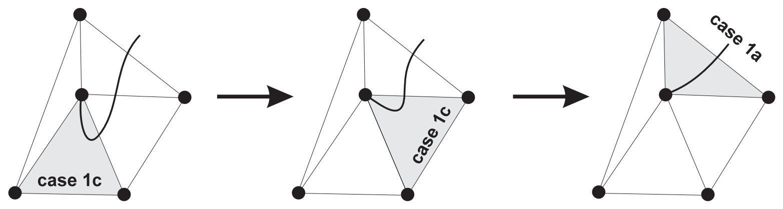

Figure 10: A Case 1c triangle transformed homotopically to a Case 1a triangle

simple set of equations.

$$\begin{aligned}\gamma + \beta &= |A| \\ \alpha + \gamma &= |B| \\ \beta + \alpha &= |C|.\end{aligned}$$

We can solve this simple set of equations and obtain

$$\begin{aligned}\alpha &= (|B| + |C| - |A|)/2 \\ \beta &= (|C| + |A| - |B|)/2 \\ \gamma &= (|A| + |B| - |C|)/2.\end{aligned}$$

Since the flow at any vertex is conserved, we have $A + B + C = 0$. The parity of $A + B + C =$ parity$(0) = \mathbf{E}$. It is straight forward to verify that parity$(|B|+|C|-|A|) =$ parity$(|C|+|A|-|B|) =$ parity$(|A| + |B| - |C|) =$ parity$(A + B + C) = \mathbf{E}$. Therefore α, β and γ are all integers. Also,

$$\begin{aligned}|A| + |B| - |C| &\geq |A + B| - |C| \\ &= |-C| - |C| \\ &= 0,\end{aligned}$$

so α, β and γ are non-negative.

It is a simple matter to find the subflows in Case 1a and Case 2.

We stitch the transformations of all triangles together to obtain the final topological routing.

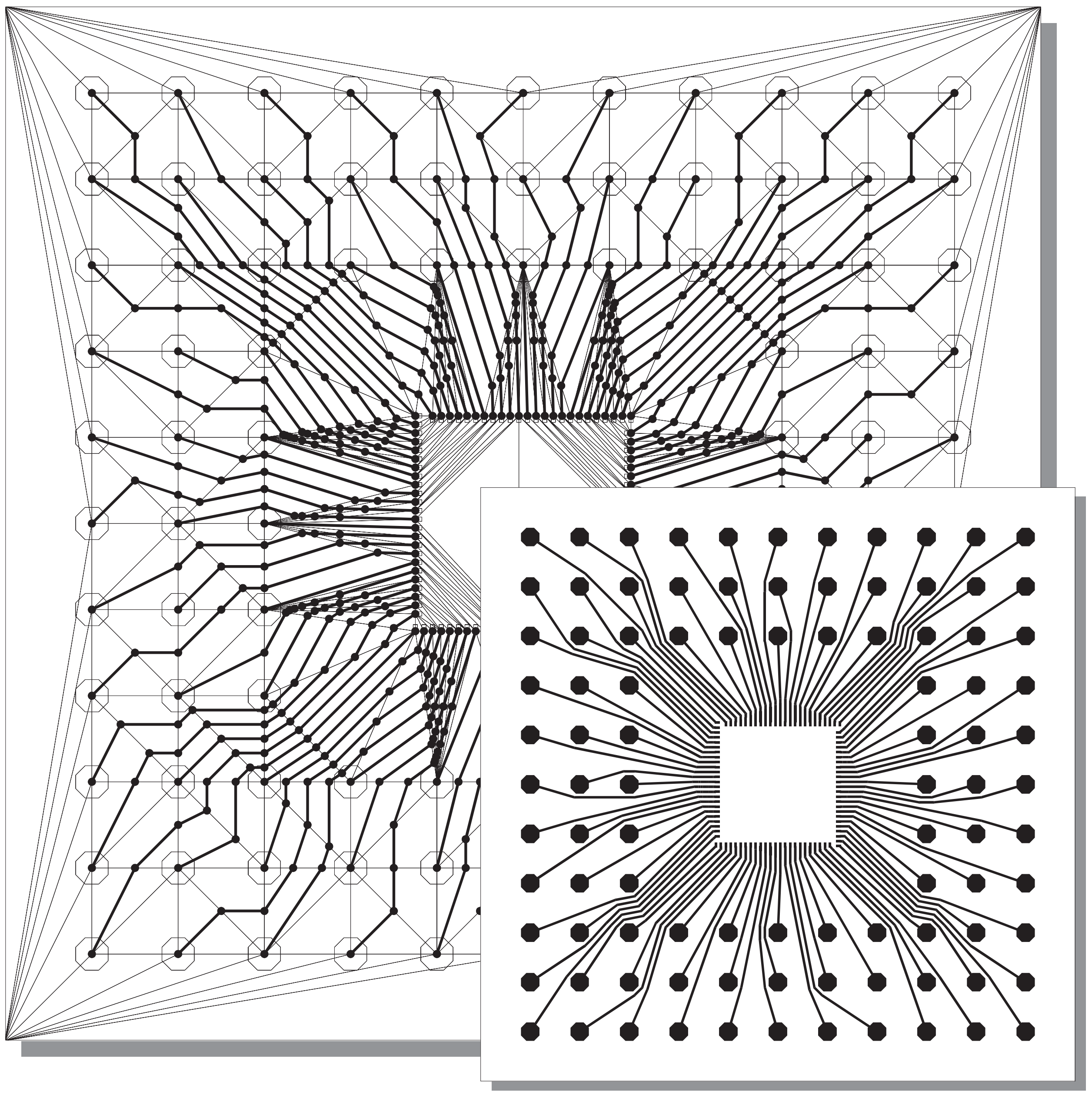

Figure 11: 96 pin PGA with the triangulation graph and wiring

5 Experimental Results

Fig. 11 shows a small PGA with its triangulation and its routing done by the flow router. After routing, the intermediate points can be relaxed using methods proposed in [12, 13, 6, 14].

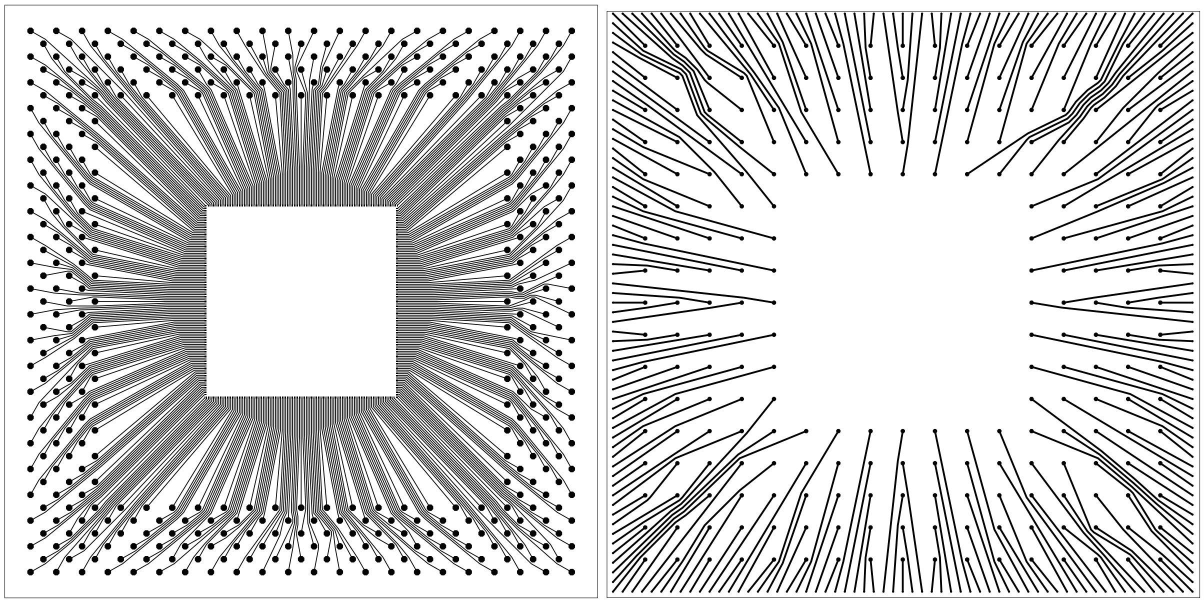

Figure 12: 444-pin PGA package and 240-pin BGA package

The router is used on a 444-pin pin grid array package with staggered pins, a 240-pin ball grid array package and a 400-connection industry example. (Fig. 12 & 13). In all examples a topological routing is created and transformed to a detailed routing with methods described by Dai *et al*[6]. All the examples are completed without design rule violations. The whole routing process including triangulation, building the routing network, running the min-cost algorithm and transform flow into topological routing takes less than 20 minutes on the probe card example on an HP9000 Model 735/99.

6 Conclusion

A large number of diverse practical routing problems in ASIC, packaging and testing can be reduced to the Planar Interchangeable Two-terminal Routing problem. In this paper, we have shown that despite the freedom of pin assignment, PI2TR is NP-complete. We developed a min-cost flow router heuristic to solve this problem. The router was applied to solve an array of routing problems in PGA, BGA and other industry examples. Experimental results show that the heuristic was efficient in computing time and produced very good results. It is a simple matter to generalize the routing network to multilayer with no vias.

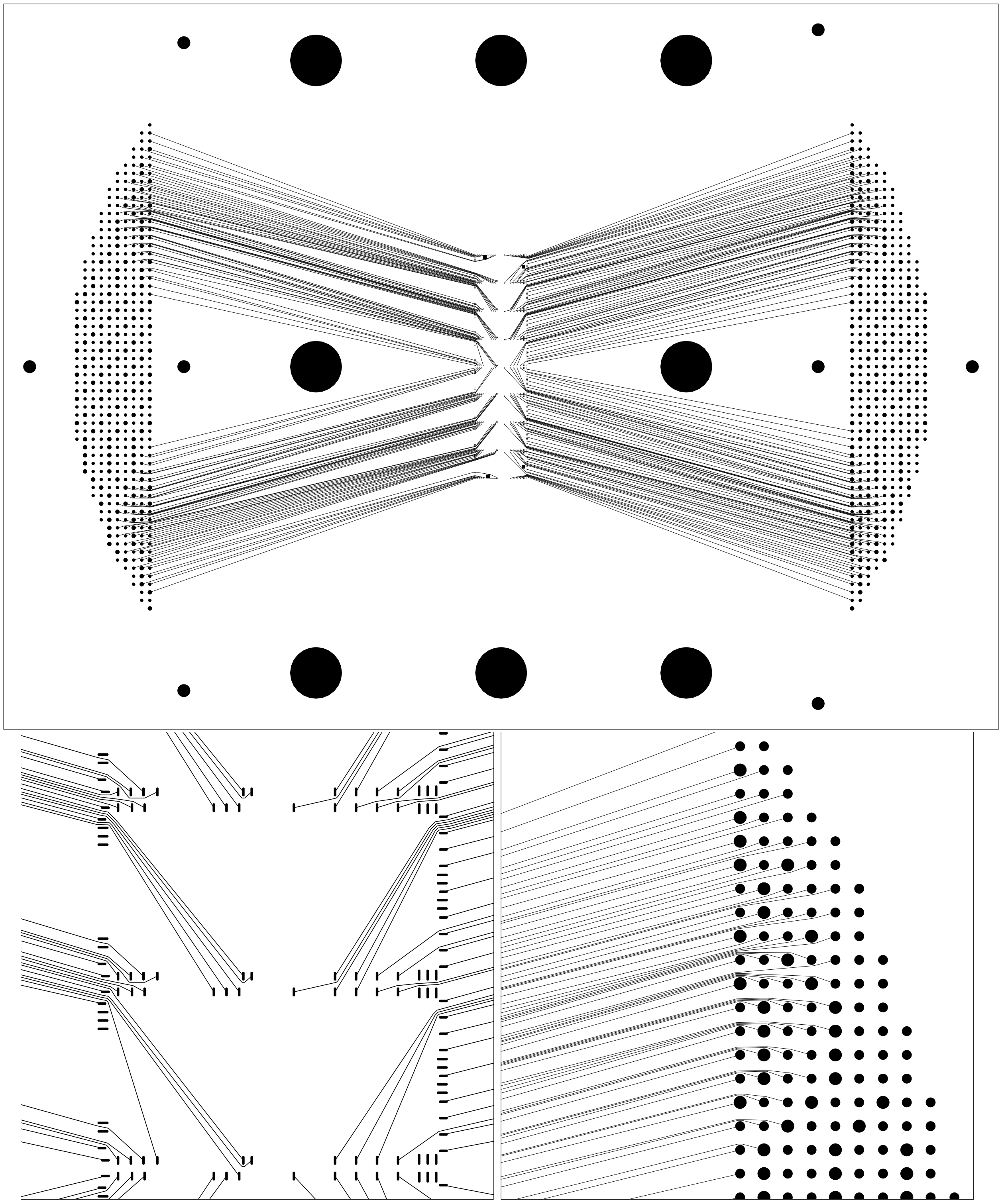

Figure 13: 400-connection test probe card

References

[1] M.-F. Yu and W. W.-M. Dai, "Pin assignment and routing on a single-layer pin grid array," in *Proc. 1st Asia and South Pacific Design Automation Conf.*, (Makuhari, Japan), pp. 203–208, IEEE Computer Society Press, June 1995.

[2] M.-F. Yu and W. W.-M. Dai, "Single-layer fanout routing and routability analysis for ball grid arrays," in *IEEE/ACM Intl. Conf. CAD-95*, (San Jose, CA), IEEE Computer Society Press, November 1995.

[3] J. Darnauer and W. W.-M. Dai, "Fast pad redistribution from periphery-io to area-io," in *Proc. IEEE Multi-Chip Module Conf.*, (Santa Cruz, CA), pp. 38–43, March 1994.

[4] C. Ying and J. Gu, "Automated pin grid array package routing on multilayer ceramic substrates," *IEEE trans. VLSI Systems*, vol. 1, no. 4, pp. 571–575, 1993.

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

[6] W. W.-M. Dai, R. Kong, and M. Sato, "Routability of a rubber-band sketch," in *Proc. 28th Design Automation Conf.*, (San Francisco, CA), pp. 45–48, IEEE Computer Society Press, 1991.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness.* San Francisco: W. H. Freeman, 1979.

[8] Y. Lu, "Dynamic constrained delaunay triangulation and application to multichip module layout," Master's thesis, University of California, Santa Cruz, 1991.

[9] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction.* New York: Springer-Verlag, 1985.

[10] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Englewood Cliffs, NJ: Prentice-Hall, Inc, 1982.

[11] R. E. Tarjan, *Data Structures and Network Algorithms.* Phildelphia, PA.: SIAM Publications, 1983.

[12] 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.

[13] W. W.-M. Dai, T. Dayan, and D. Staepelaere, “Topological routing in surf: Generating a rubber-band sketch,” in *Proc. 28th Design Automation Conf.*, (San Francisco, CA), pp. 39–44, IEEE Computer Society Press, June 1991.

[14] W. W.-M. Dai, R. Kong, J. Jue, and M. Sato, “Rubber band routing and dynamic data representation,” in *Proc. 1990 Int'l Conf. on CAD*, (San Jose, CA), pp. 52–55, IEEE Computer Society, November 1990.