Interchangeable Pin Routing with Application to Package Layout

Man-Fai Yu
Joel Darnauer
Wayne Wei-Ming Dai

UCSC-CRL-96-10
April 25, 1996

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

ABSTRACT

Many practical routing problems such as BGA, PGA, pin redistribution and test fixture routing involve routing with interchangeable pins. These routing problems, especially package layout, is becoming more difficult to do manually due to increasing speed and I/O. Currently, no commercial or university router is available for this task. In this paper, we unify these different problems as instances of Interchangeable Pin Routing (IPR) problem. We show that this problem is NP-complete. We formulate the problem as flows in a routing network on a triangulation instead of grids and developed a min-cost max-flow heuristic considering only the most important cuts in the design. The heuristic is extended to multiple layers and handles prerouted nets. It can accommodate all-angle, octilinear or rectilinear metric. Experiments showed that the heuristic is very effective on most practical examples. It successfully routed an industry design with 4000 interchangeable pins without manual intervention.

Keywords: PGA, BGA, package layout, planar routing, routability, network flow, interchangeable pins

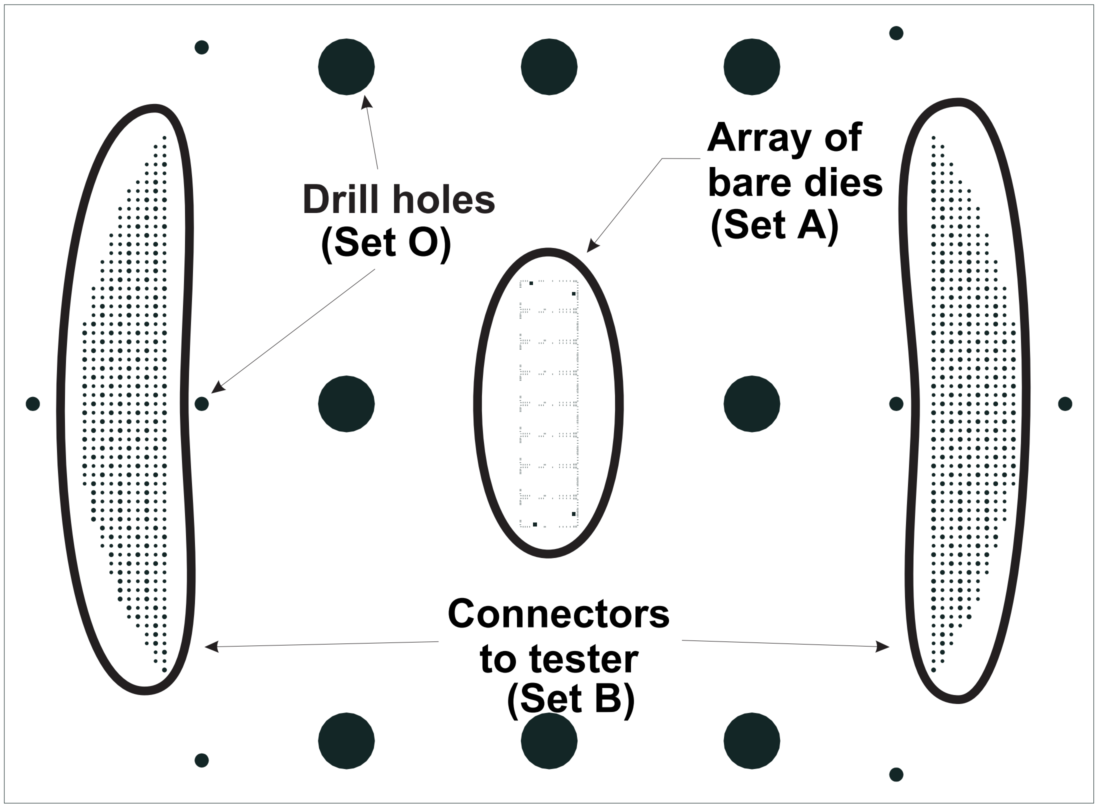

Figure 1: An unrouted example of routing with exchangeable pins. Each pad in the center must be connected to one pad on the edge connector.

1 Introduction

Package layout has been a missing link in design automation. It is done manually because packages has been simple enough and the layout tools cannot produce all-angle, non-uniform width wires. Nowadays array I/O packages like Ball Grid Arrays (BGA) and Pin Grid Arrays (PGA) are complex structures with multiple routing layers, power and ground planes and decoupling capacitors built in. High I/O count and high-speed performance requirements make manual design more costly and time-consuming. Although some package routers exist[18, 19], they have a number of fatal limitations that prevent them from being adopted by industry for practical use. This paper proposes a more general router that does not have these limitations and is suitable for a wide variety of applications, ranging from BGA and PGA routing to test probe card design.

In BGA or PGA package design, we wish to connect each chip pad to a single package pin, but we may not 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 I/O pads on the periphery. Figure 1 shows a test probe card design. We want to connect all pins in set A to set B while avoiding all obstacles. We do not care which pin in B is connected by a pin in A as long as there is *some* pin.

Conventional routers cannot address these problems because they require a predefined pin assignment in the form of a netlist, before the routing process can be started. In addition to adding an extra step to the design process, the choice of pin assignment is critical to the 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[18, 19, 8, 17] have been developed that take advantage of special geometries and symmetries of their respective problems and the freedom of interchangeable pins. Although these methods are exceptionally efficient, they have a number of limitations:

Geometry dependent They rely on the symmetry of the arrays and rings to generate solutions and cannot cope with missing, skewed, off-grid or arbitrarily placed pads.

Routability guarantee These algorithms cannot always tell if they have generated a routable solution, and have no strategy for changing an unroutable solution into a routable solution.

Unequal sets They require the two pin sets to be the same size. In actual packages the number of pins available may be larger than the number of chip I/O. The router should have the freedom to select which pins to use.

Obstacles They do not take into account the presence of obstacles.

Multilayer It is not easy to extend Yu and Dai's[18, 19] routers to multiple layers.

Prerouting These routers cannot accept prerouted wires. Prerouting is very important in package design because the designer wants to be able to route all the critical nets before other non-critical nets.

Responding to these limitations, we propose a general problem formulation called *Planar Two-Terminal Interchangeable Pin Routing (P2TIR)*. We showed that although we can take advantage of the freedom of interchangeable pins, P2TIR is NP-complete so no polynomial-time algorithm is likely to exist that guarantees routability. Despite of its NP-completeness, we offer an efficient heuristic which is *independent* of pin placement, handles *unequal sets and obstacles*, can be extended to *multiple layers* and observes all *prerouting*. Experiments showed that this heuristic, which we called the *flow router*, is very effective on practical examples, including some provided by the industry. We also showed that the router scales well to handle production-sized jobs from industry. The router successfully completed our largest example of 4000 interchangeable pins without manual intervention. No commercial or university router is capable of handling interchangeable pin problems of this size.

Cho and Sarrafzadeh[4, 3, 5] formulated the Pin Redistribution Problem for routing redistribution layers in Ceramic Multichip Modules (MCM-C). This problem on a single layer is similar to P2TIR except that all objects are on a fixed grid and all wires are Manhattan. Chang *et al*[1] uses a flow approach similar to ours for solving the Pin Redistribution Problem. Since the problem formulation is less general it is not easy to generalize the solution to other technologies. In addition, every grid cell corresponds to a node in their flow graph so the number of nodes in the graph is $O(m^2)$ where m is the dimension of the design. For a test probe card, the length of the card is in the order of inches and the chips to be tested has pad pitch in the order of several mils so grid-based approach is practically impossible. In this work we consider the most general case of *all-angle* wiring and our solution works for Euclidean, octilinear or rectilinear metric. Our routing network is not based on grids but on triangulation, which scales linearly with the number of pins. Therefore our solution is more technology-independent and computationally more efficient.

1.1 Formal Problem Definition

The fundamental features of P2TIR 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 all obstacles, as well as the particular design rules permitted by the wiring. Accordingly, we define an instance of the Planar 2-Terminal Interchangeable Pin Routing (P2TIR) problem as follows:

Definition 1 Planar 2-Terminal Interchangeable Pin Routing

Instance *A 6-tuple of* (b, A, B, O, w, s) *where:*

> b *is a polygon representing the routing area boundary.*
>
> 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* $|A| \leq |B|$.

Output *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 *avoiding all obstacles in* O *and obey the width and spacing rules.*

Maley[12] showed that the routability of topological routing (or homotopic routing) can be determined in polynomial time. A topological routing of a 2-terminal net is the equivalence class of all detailed routing of the net under homotopic transformation. A topological routing T is routable if and only if there exists a detailed routing in T that satisfies all design rules. Maley also showed that a topological routing is routable if the total number of wires flowing through straight 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[7], we will consider a routable topological routing a solution for P2TIR.

2 Network flow formulation of P2TIR

In this section we develop a network flow formulation for P2TIR. We showed that any flow assignment can be transformed to a topological routing.

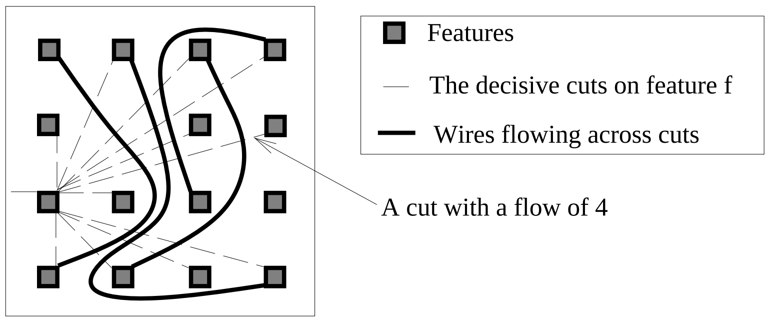

Figure 2: A topological routing with some cuts and flows illustrated

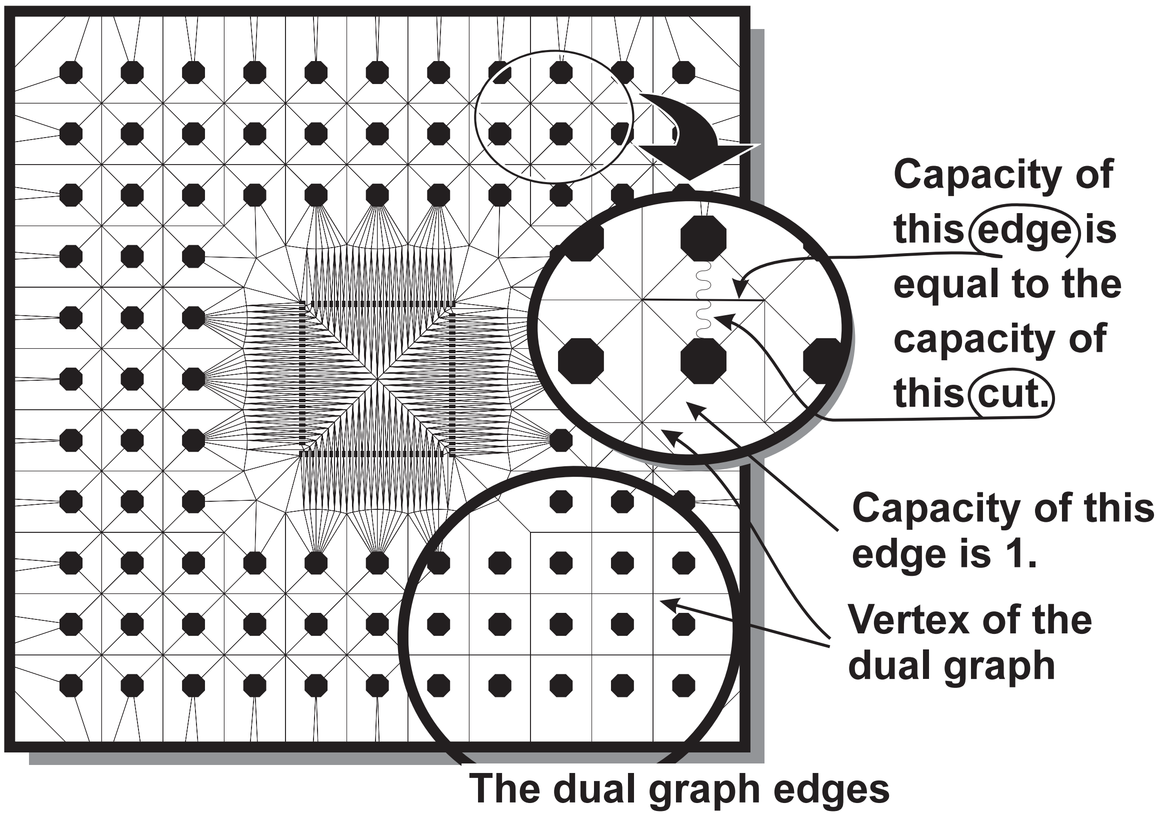
Capacity of this edge is equal to the capacity of this cut.
Capacity of this edge is 1.
Vertex of the dual graph
The dual graph edges

Figure 3: The Routing Network of a PGA package

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.

The routing network $T(V, E, s, t)$ is a directed graph with a source s and a sink t. We first shrink each pin in sets A and B and each obstacle in O into a point. Each representing point has to be within the boundary of its object. Let these sets of points be $\tilde{A}$, $\tilde{B}$ and $\tilde{O}$ respectively. Then we build a Delaunay triangulation D on $\tilde{A} \cup \tilde{B} \cup \tilde{O} \cup b$, where b is the set of points of the bounding polygon. From the triangulation graph, we define T as follows:

Definition 2 *If Δ is the dual of D, a* Routing Network $T(V, E, s, t)$ *is a network where $V = \Delta \cup \tilde{A} \cup \tilde{B}$. E consists of the following arcs:*

- *A pair of opposite arcs for each edge in Δ.*
- *A pair of opposite arcs between each point in $\tilde{A} \cup \tilde{B}$ and each of its incident triangles.*

Each arc has a capacity and a non-negative cost.

The arcs originating from a point in $\tilde{A}$ (the source set) have unit capacities and arcs to a point in $\tilde{A}$ have zero capacities. The opposite is true for $\tilde{B}$ (the sink set). Finally, the pair of arcs connecting two triangles have capacities equal to the maximum number of wires that can cross between the corresponding pins or obstacles in both directions. The capacities should account for the finite size of pins or obstacles. The supersource s has an outgoing arc to each vertex in $\tilde{A}$ with capacity equal to 1 and cost equal to 0. Similarly, the supersink t has an incoming arc from each vertex in $\tilde{B}$.

Note that the vertices in Δ are triangles. Since there are many possible triangulations for each instance of P2TIR, 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. Any flow assignment on this network has the following properties.

- The flow entering a vertex in $\tilde{A}$ must be -1.
- The flow entering a vertex in $\tilde{B}$ must be either 0 or $+1$. (Some pins 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 3 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[12] showed that only a certain set of cuts, the *decisive cutset*, needs to be checked for a topological routing to determine its routability. If there are N objects, 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 but not all* 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 obviously possible 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. Appendix A showed that P2TIR is in fact NP-complete so no polynomial time algorithm that finds a routable topological routing is likely to exist.

3 The Flow Router

In this section we describe in detail the **flow router** as a heuristic to solve P2TIR. 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.

3.1 Building the routing network

We used the incremental Delaunay triangulation described by Lu[11] to construct the triangulation, although any other algorithms are equally good. The Delaunay triangulation of $N = |\tilde{A} \cup \tilde{B} \cup \tilde{O}|$ points can be constructed in $O(N \log N)$ time[14]. 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.

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 4: Algorithm buildup

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.

3.2 The Min-cost Max-flow Algorithm

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[13]. Informally, we try to find a minimum total cost assignment of flows for

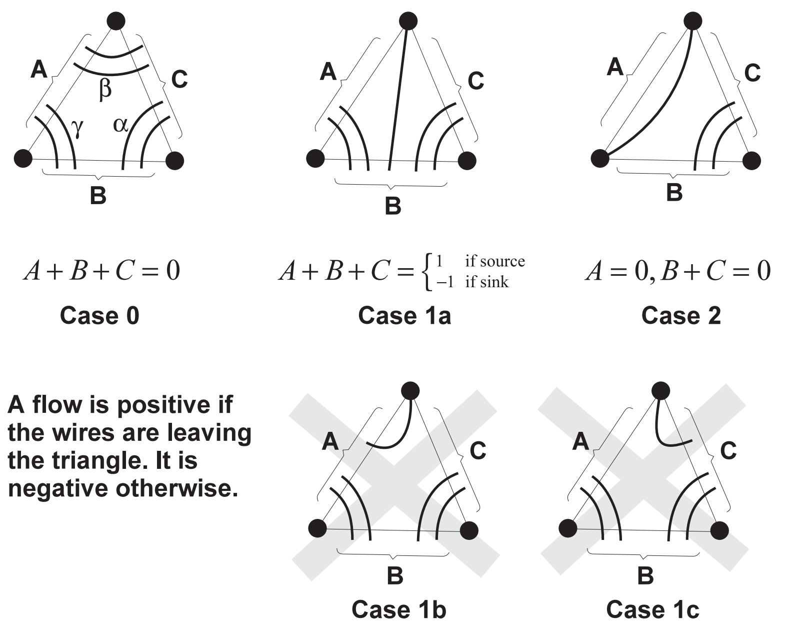
A
B
C
β
γ
α
$A+B+C=0$
Case 0
$A+B+C=\begin{cases}1 & \text{if source}\\ -1 & \text{if sink}\end{cases}$
Case 1a
$A=0, B+C=0$
Case 2
A flow is positive if the wires are leaving the triangle. It is negative otherwise.
Case 1b
Case 1c

Figure 5: Three cases of mapping flows in a triangle to a topological routing

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[16]. This algorithm runs in $O(|V||E|)$ time. Note that the flow on an edge can be positive or negative. The direction of the flow is always from the source to the sink. Therefore the run time of the flow assignment algorithm is $O(|V|^3)$ since the number of edges in the triangulation is linearly proportional to the number of vertices. $V = A \cup B \cup O$. Fig. 4 shows both algorithms.

3.3 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. 5 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 simplify 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. 6 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

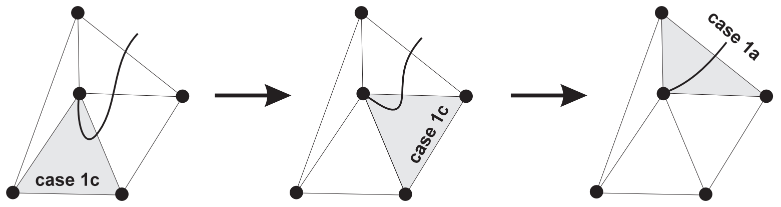

Figure 6: A Case 1c triangle transformed homotopically to a Case 1a triangle

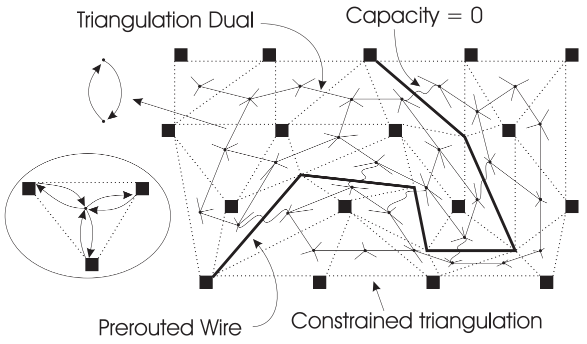

Figure 7: A prerouted wire as constrained edges and its (partial) routing network

following simple set of equations.

$$\gamma + \beta = |A| \quad \alpha + \gamma = |B| \quad \beta + \alpha = |C|.$$

We can solve this simple set of equations and obtain

$$\alpha = (|B| + |C| - |A|)/2 \quad \beta = (|C| + |A| - |B|)/2 \quad \gamma = (|A| + |B| - |C|)/2.$$

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

$$|A| + |B| - |C| \geq |A + B| - |C| = |-C| - |C| = 0,$$

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.

3.4 Handling Prerouted Nets and Extension to Multiple Layers

We handle prerouted nets by constrained edges. We assume that each prerouted net is piecewise-linear. Then we embed each wire as a set of constrained edges into the triangulation. We use the set of algorithms reported by Lu[11] for building and maintaining a Constrained Delaunay Triangulation (CDT). The capacities of corresponding routing network arcs are set to 0 so that wires cannot intersect each other. In effect the prerouted wires become barriers of flows. Since the prerouted wires are directly embedded into the triangulation, their exact shape affects the flow solution. Fig. 7 shows a prerouted wire embedded in the triangulation and the resulting dual.

For multiple layers, we build triangulation and create the dual graphs on each layer as in the case of single layers. Then we combine these graphs together. Each pin is now a pin stack that spans contiguous, but not necessarily all, layers. So we extend the definition of the routing network as follows. We first choose a point to represent each pin stack. We call, as before, these sets of points $\tilde{A}, \tilde{B}$ and $\tilde{O}$ respectively for sets A, B and O. Let D_i be the triangulation of the ith layer, for $i = 1, \ldots, N$ layers.

Definition 3 *If* Δ_i *is the dual of* D_i*, a* Routing Network $T(V, E, s, t)$ *is a network where* $V = \tilde{A} \cup \tilde{B} \cup \sum_{i=1}^{N} \Delta_i$*. E consists of the following arcs:*

- *A pair of opposite arcs for each edge in* Δ_i*.*
- *A pair of opposite arcs between each point in* $\tilde{A} \cup \tilde{B}$ *and each of its incident triangles on layer* i *if the corresponding pin stack has a pad on layer* i*.*

Each arc has a capacity and a non-negative cost.

Note that each pin stack corresponds to only one (source or sink) vertex in the routing network. But each sink or source vertex has a pair of arcs to their incident triangles on *all* layers. The capacity of each cut is set according to the specific design rules of the layer the cut belongs to. Therefore each layer can have different design rules. The cost of each arc can also be adjusted so that some layer has higher cost per unit wire length than others. The min-cost algorithm will find a solution with minimum weighted wire length if one exists.

4 Experimental Results

Fig. 8 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 by Dai *et al.* [15, 7, 6].

The first non-trivial example is a single-layer 444 pin PGA package with staggered pins. Yu and Dai[18] cannot handle staggered pins. Another example is a two layer 280 pin BGA package. Again the router of Yu and Dai[19] only handles a single layer. The router automatically distributes the wires between the two layers and can accommodate different design rules of each layer (Fig. 9). To show that the router does not assume any pin placement, we routed a 400 pin test probe card provided by industry. The card connects chip I/Os at the center to two arrays of pads at the periphery. The router has to avoid prerouted nets and obstacles such as drill holes and other reserved areas (Fig. 10). Our last example shows that the router can handle large production-scale jobs. We use the same basic probe card layout and randomly generated 2148 pins. The flow router completed the routing on two layers with less than 10 design rule violations in 5 hours of CPU time on an HP9000 Model 735/99 workstation (Fig. 11). The design rule violations were fixed easily by manually editing the routing using Surf[15] in less than 30 minutes. So far no other routers, commercial or university, is capable of handling this. The largest example completed by the flow router was a 4000 pin test probe card. The routing is completed without manual intervention with only 140 design rule violations.

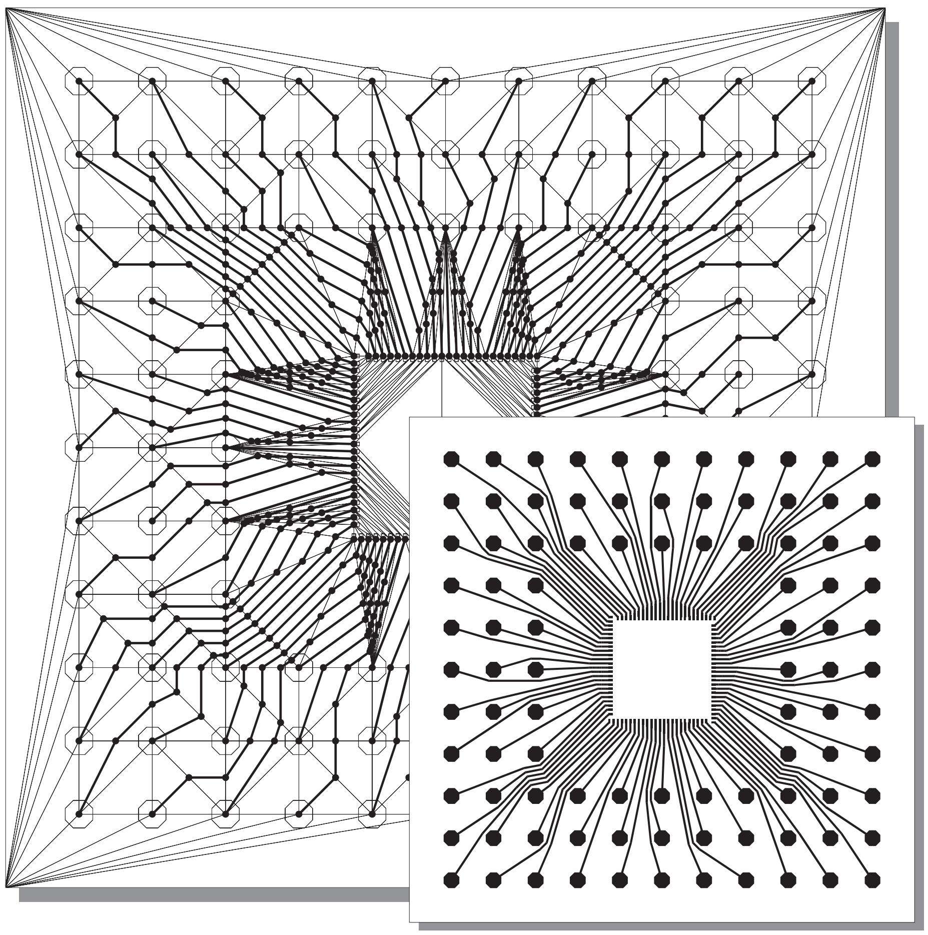

Figure 8: 96 pin PGA with the triangulation graph and wiring

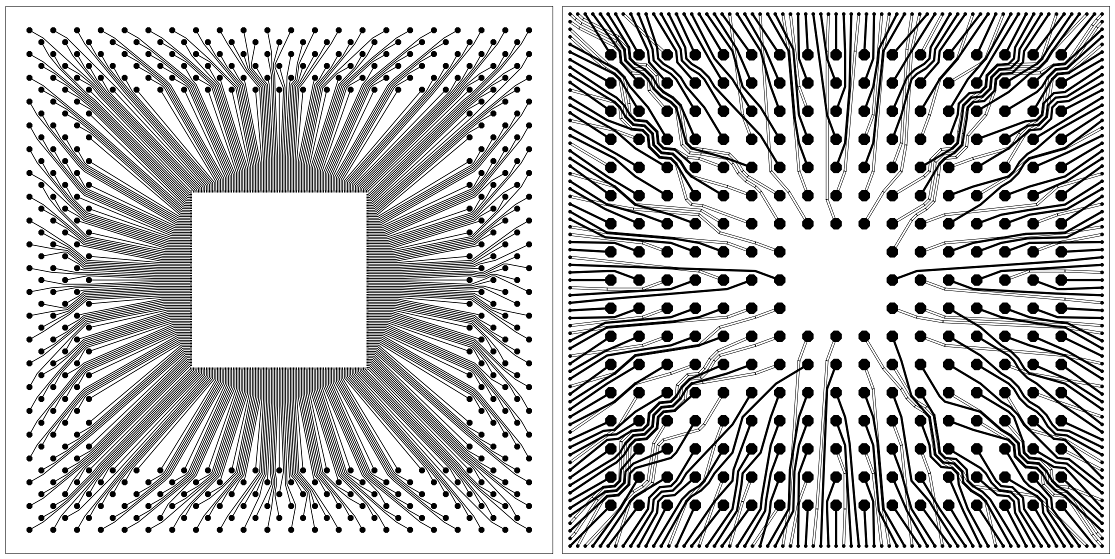

Figure 9: A 444 staggered pin PGA package and a two layer 280-pin BGA package

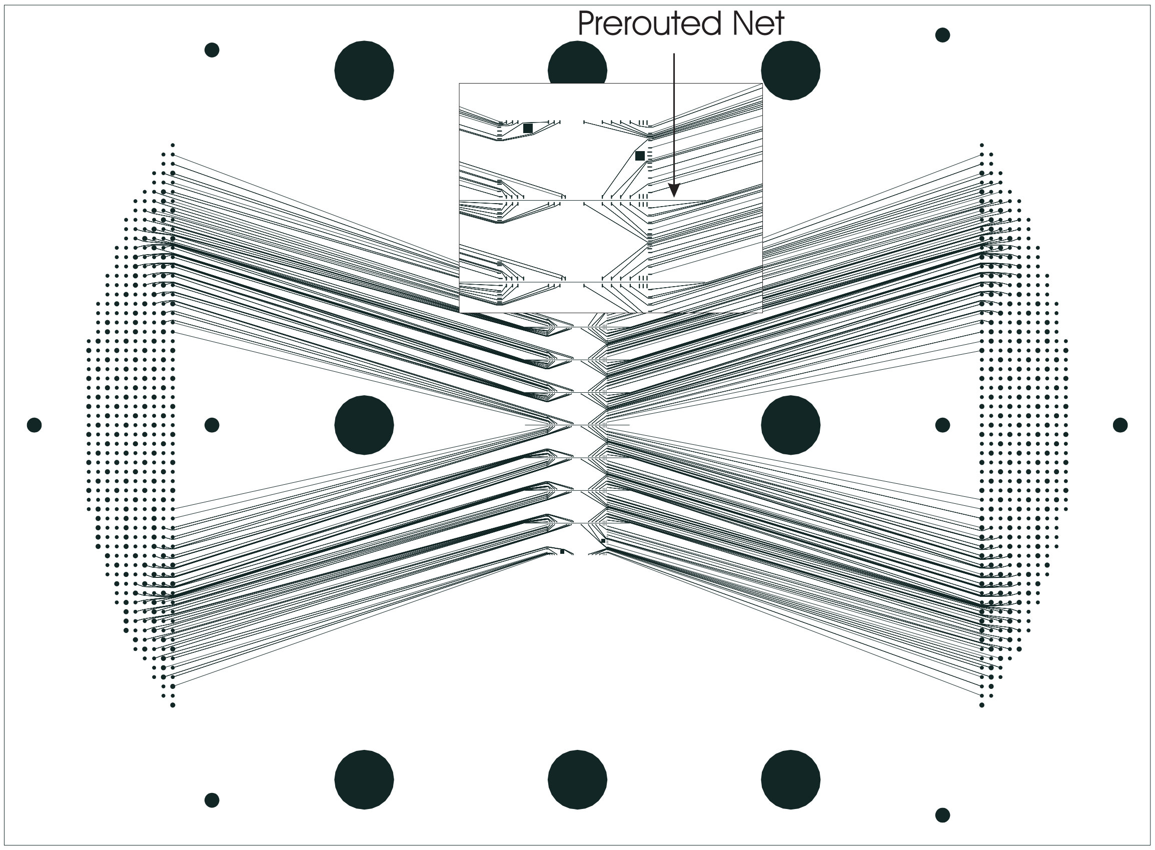

Figure 10: 400 connection test probe card with prerouted nets. These nets constrained the routing so that wires to each chip is grouped together.

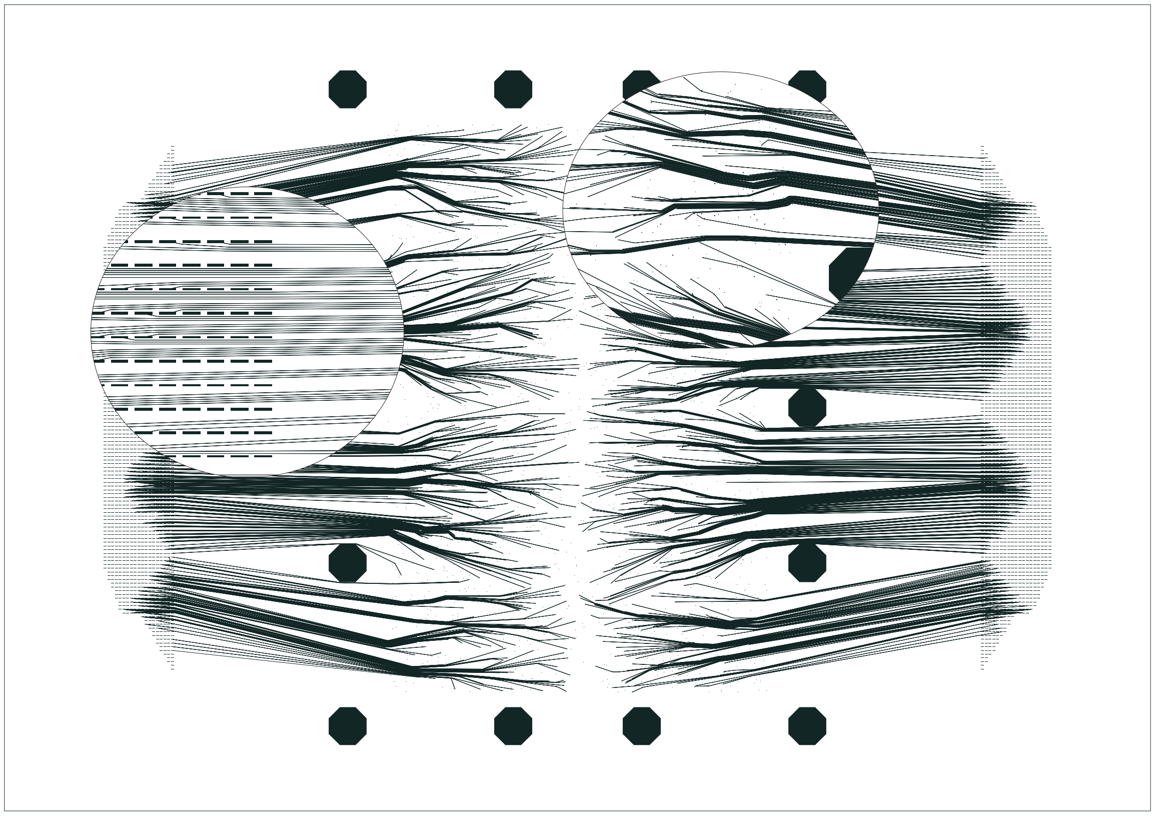

Figure 11: Two layer test probe card with 2148 randomly generated pins (one layer shown)

In all examples a topological routing is created and transformed to a detailed routing with methods described by Dai *et al*[7]. The whole routing process includes triangulation, building the routing network, running the min-cost algorithm and transform flow into topological routing.

5 Conclusion

A large number of diverse practical routing problems in ASIC, packaging and testing can be reduced to the Planar Two-terminal Interchangeable Pin Routing problem. In this paper, we have shown that despite the freedom of pin assignment, P2TIR 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 test probe cards. The router runs on multiple layers and handles prerouted nets. Experiments show that the heuristic is efficient in computing time and produced very good results.

A P2TIR is NP-complete

P2TIR can be considered the *weakest* or the most restricted routing problem not only because it is planar and all nets are two-terminal, but also because pins are interchangeable so any solution can take advantage of the flexibility of pin assignment. We will show that

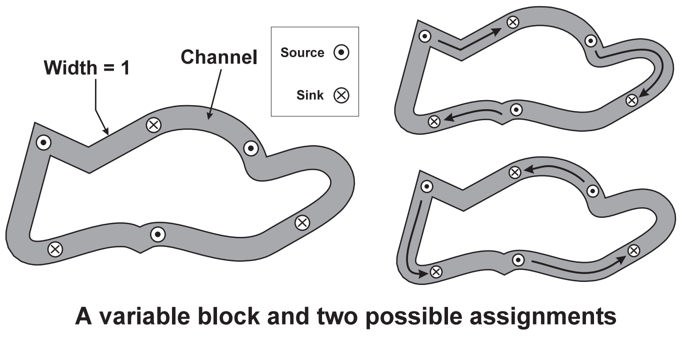

Figure 12: A boolean variable

determining the existence of a routable solution of P2TIR is still NP-complete.

Problem 1 (Existence of a Planar 2-Terminal Interchangeable Pin Routing (EP2TIR))

INSTANCE Two sets of pins A and B, a design rule $R = \{w, s\}$ where w is the minimum width of a wire and s is the minimum spacing between two wires or between a wire and a pin, a set of obstacles O.

OUTPUT Answer to the question "Is there an embedding of wires that connect pins between A and B satisfying R and avoiding all the obstacles in O?"

Maley[12] showed that given a planar 2-terminal topological routing, a polynomial-time algorithm exists to verify its routability. He provided an algorithm that runs in $O(n^2 \log n)$ time where n is the sum of the number of obstacles and wires. Chen and Lee[2] gave a plane sweep algorithm that produced a detailed rectilinear routing in $O(|F||W|)$ time where F is the set of features (pins and obstacles) and W is the set of wires. Surf[7] has an incremental design rule checker. Therefore EP2TIR $\in$ NP. We will reduce PLANAR 3-SAT to EP2TIR. PLANAR 3-SAT is defined as follows:

Problem 2 (PLANAR 3-SAT)

INSTANCE A set of boolean variables V, a set of clauses $C = \{(v_1 + v_2 + v_3) | v_i \in V \text{ or } \overline{v_i} \in V, i = 1, 2, 3\}$, a planar bipartite graph $G = (V \cup C, E)$ such that an edge $(v, c) \in E$ if and only if v or $\overline{v}$ appears in c.

OUTPUT The answer to the question "Is there an assignment $A : V \rightarrow \{\mathbf{T}, \mathbf{F}\}$ such that all clauses in C evaluate to $\mathbf{T}$?"

Lichtenstein[10] showed that PLANAR 3-SAT is NP-complete. In the following we will construct several components and describe a procedure to construct a layout instance given a PLANAR 3-SAT instance.

For convenience we call the terminals in the smaller set A in an instance of EP2TIR the "source" terminals and those in B as the "sink" terminals. Figure 12 shows a routing channel loop with an alternate sequence of source terminals and sink terminals. The design rule is such that only one wire can be routed in the channel. The sources and sinks are

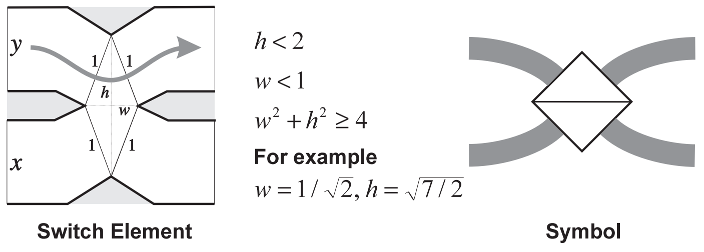

Figure 13: A basic switch element

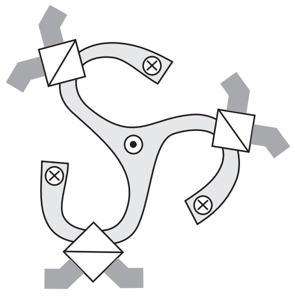

Figure 14: A clause

always in pairs so two and only two states are possible. We can arbitrarily assign one state **T** and the other **F**. We call this loop a "variable" block.

Figure 13 shows a "switch" element. Two routing channels meet at a junction where only one wire can pass. If there is a wire in the lower channel, it is impossible to have a wire in the upper channel and vice versa. The width of the junction is not enough to route one wire through so a wire cannot start in the lower channel and ends in the upper channel and vice versa. Using this element we can build a "clause" block (Figure 14). A clause block has one source and three sinks connected by routing channels. On each channel between the source and sink there is a switch. The switch joins with either an unnegated or a negated channel of a variable block. If the variable appears unnegated in the clause, the switch connects with the *negated* channel of the variable loop. If the variable appears negated in the clause, the switch connects with the *unnegated* channel of the loop. It is easy to see that for the structure to have a feasible flow, at least one switch has to allow a route, requiring a truth assignment from the variables.

Now we will prove the following.

1. An instance L of EP2TIR can be constructed in polynomial time given an instance S of PLANAR 3-SAT.

2. A solution exists for L if and only if S is satisfiable.

Proof: For each S, we can construct a corresponding layout instance L by using variable and clause blocks. Hopcroft and Tarjan showed that an embedding for any planar graph can be found in polynomial time[9]. From this embedding we replace each clause vertex by

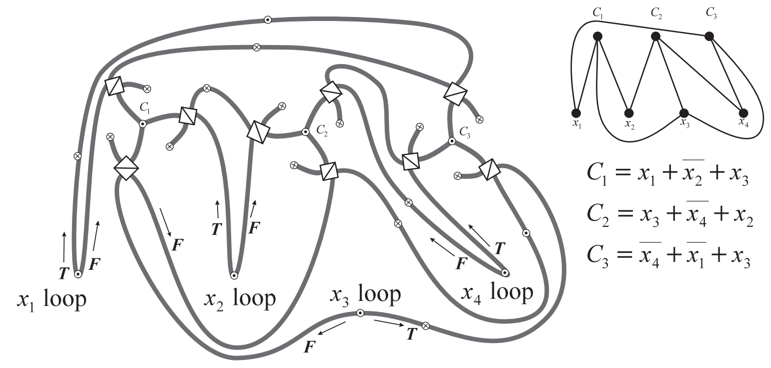
C1
C2
C3
T
F
x1 loop
x2 loop
x3 loop
x4 loop
x1
x2
x3
x4
$C_1 = x_1 + \overline{x_2} + x_3$
$C_2 = x_3 + \overline{x_4} + x_2$
$C_3 = \overline{x_4} + \overline{x_1} + x_3$

Figure 15: EP2TIR instance for $(x_1 + \overline{x_2} + x_3)(x_3 + \overline{x_4} + x_2)(\overline{x_4} + \overline{x_1} + x_3)$

a clause block and each variable vertex by a variable loop. For each edge from a variable to a clause, we put in a pair of parallel channels that connect to the clause block at a switch. On the other end, the variable loop is cut and joined with the pair of channels. We place pairs of sources or sinks into the channel so that the variable appears negated or unnegated at the clause. The total time is proportional to sum of the number of clauses, variables and edges. □

PROOF: If S is satisfiable, there exists a truth assignment $A : V \rightarrow \{\mathbf{T}, \mathbf{F}\}$ such that all clauses in C evaluates to $\mathbf{T}$. We set the routing in each variable block according to the assignment of the variable. Since A is a truth assignment, at least one literal in each clause is not false so the corresponding switch element in the clause block allows a route from the source to a sink. Therefore L has a routable solution.

Consider a routable solution of L. Each clause block has a route from its source to one of its sinks. If the switch on this route joins with the variable block that represents some variable v, v can only be in one state because the switch blocks either a channel that represents v or a $\overline{v}$ channel. Since this is a routable solution, switches along the loop will be consistent with each other, i.e. either all switches block v channels or block $\overline{v}$ channels but not both. It is possible that in some variable blocks none of the channels are blocked. In this case we arbitrarily assign $\mathbf{T}$ to the variable. We can read off the truth assignment by examining the state of each variable block in polynomial time. □

Figure 15 shows the corresponding routing instance for the boolean formula $(x_1 + \overline{x_2} + x_3)(x_3 + \overline{x_4} + x_2)(\overline{x_4} + \overline{x_1} + x_3)$, where $C_1 = (x_1 + \overline{x_2} + x_3)$, $C_2 = (x_3 + \overline{x_4} + x_2)$ and $C_3 = (\overline{x_4} + \overline{x_1} + x_3)$.

References

[1] CHANG, D., GONZALEZ, T. F., AND IBARRA, O. H. A flow based approach to the pin redistribution problem for multi-chip modules. In *Proc. 4th Great Lakes Symposium on VLSI* (University of Notre Dame, IN, March 1994), IEEE Computer Society, pp. 114–119.

[2] CHEN, H.-F. S., AND LEE, D. T. A faster algorithm for rubber-band equivalent transformation for planar VLSI layouts. *IEEE Trans. Computer-aided Design 15*, 2

(February 1996), 217–227.

[3] CHO, J. D., LIAO, K.-F., RAJIE, S., AND SARRAFZADEH, M. *M^2R: Multilayer routing algorithm for high-performance MCMs.* *IEEE Trans. Circuits and Systems I 41*, 4 (April 1994), 253–265.

[4] CHO, J. D., AND SARRAFZADEH, M. The pin redistribution problem in multi-chip modules. In *Proc. 4th IEEE Intl. ASIC Conf. Exhib.* (Rochester, NY, September 1991), IEEE, pp. P9–2.1–P9–2.4.

[5] CHO, J. D., AND SARRAFZADEH, M. An optimum pin redistribution for multichip modules. In *Proc. IEEE Multichip Module Conf.* (Santa Cruz, CA, Febuary 1996), IEEE, pp. 111–116.

[6] DAI, W. W.-M., KONG, R., JUE, J., AND SATO, M. Rubber band routing and dynamic data representation. In *Proc. Intl. Conf. Computer-aided Design* (San Jose, CA, November 1990), IEEE Computer Society, pp. 52–55.

[7] DAI, W. W.-M., KONG, R., AND SATO, M. Routability of a rubber-band sketch. In *Proc. 28th IEEE/ACM Design Automation Conf.* (San Francisco, CA, 1991), IEEE Computer Society Press, pp. 45–48.

[8] DARNAUER, J., AND DAI, W. W.-M. Fast pad redistribution from periphery-IO to area-IO. In *Proc. IEEE Multichip Module Conf.* (Santa Cruz, CA, March 1994), pp. 38–43.

[9] HOPCROFT, J., AND TARJAN, R. E. Efficient planarity testing. *J. ACM 21*, 4 (October 1974), 549–568.

[10] LICHTENSTEIN, D. A technique for proving NP-completeness results on planar graphs. Master's thesis, University of California, Berkeley, Berkeley, CA, December 1977.

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

[12] MALEY, F. M. *Single-layer wire routing and compaction.* MIT Press, Cambridge, MA, 1990.

[13] PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc, Englewood Cliffs, NJ, 1982.

[14] PREPARATA, F. P., AND SHAMOS, M. I. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[15] STAEPELAERE, D., JUE, J., DAYAN, T., AND DAI, W. W.-M. Surf: A rubber-band routing system for multichip modules. *IEEE Design and Test of Computers* (December 1993).

[16] TARJAN, R. E. *Data Structures and Network Algorithms.* SIAM Publications, Phildelphia, PA., 1983.

[17] YING, C., AND GU, J. Automated pin grid array package routing on multilayer ceramic substrates. *IEEE trans. VLSI Systems 4*, 1 (1993), 571–575.

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

[19] Yu, M.-F., and Dai, W. W.-M. Single-layer fanout routing and routability analysis for ball grid arrays. In *Proc. Intl. Conf. Computer-aided Design* (San Jose, CA, November 1995), IEEE, pp. 581–586.