January 16, 1997

Constructive Algorithms Based on Graph Minors[*◇]

Rajeev Govindan[†], Michael A. Langston[†] and Siddharthan Ramachandramurthi[‡]

Abstract

We consider the development of practical algorithms based on the theory of graph minors. Although an exact decision algorithm using this approach would generally require testing to ensure the absence of a prohibitively large number of obstructions, approximate algorithms can be designed that test for only a few obstructions. Efficient self-reduction strategies can then be incorporated to approximate a solution to the problem at hand.

In this paper, we investigate a prototypical problem, that of finding a three-track gate matrix layout for VLSI circuits. We design a streamlined test for a half dozen of the densest obstructions, thereby approximating an exact algorithm that would require over one hundred such tests, many of which appear very difficult.

In this effort, we have also built a software package to automate the use of our tools. Experimental results obtained using this package demonstrate that it is extremely fast and suggest that, despite its theoretical limitations, its results are correct most of the time.

---

* This research has been funded in part by the National Science Foundation under grant NSF-BIR-9318160 and by the Office of Naval Research under contract N00014–90–J–1855.

◇ A preliminary version of part of this paper was presented at the International Symposium on VLSI, held in Bombay, India, in January, 1993.

† Department of Computer Science, University of Tennessee, Knoxville, TN 37996, USA.

‡ LSI Logic Corporation, Waltham, MA 02154, USA.

1 Introduction

Although advances in the field of graph minors have yielded powerful non-constructive tools to tackle many problems, algorithms derived using this approach have remained impractical. The main feature of these algorithms is their reliance on a large but finite set of forbidden structures known as "obstructions." In this paper, we study the practical potential of these novel techniques, using the well-known Gate Matrix Layout problem as an example.

To put this study in context, consider the general layout process of VLSI circuit design. The input consists of a description of a circuit in terms of its components and their interconnections. The goal is to obtain a layout of the circuit within the limits of the fabrication technology. Various criteria, including chip area, I/O distribution, interconnection length and so forth affect the cost and performance of the realized circuit. Naturally, one seeks to optimize a layout, that is, minimize cost while maximizing performance.

With the advent of computer aided design, automatic design tools have grown in popularity and are now the mainstay of circuit designers. Thus there is great emphasis on developing better layout algorithms. Alas, virtually all versions of the layout problem are $\mathcal{NP}$-Hard [Le]. Fortunately, however, we are sometimes able to take advantage of resource limitations. These may take on many forms, such as bounds on wire length, limits on chip size, constraints on signal propagation time, ceilings on pin counts, etc. Accordingly, it is often reasonable to focus attention on a fixed-parameter version of the layout problem that, unlike the general problem, is known to be solvable in polynomial time.

This is the approach we explore here. In the next three sections, we develop requisite background on Gate Matrix Layout, problem transformations and graph minors. Our main contribution is contained in Section 5, where we develop fast tests for a handful of obstructions to this layout problem. In the last three sections, we address the use of these algorithms, present experimental results and summarize our work.

2 Gate Matrix Layout

We start with a statement of the problem and a survey of its history.

2.1 Problem Statement

The gate matrix layout style was introduced by Lopez and Law [LL] in 1980 as an elegant way to layout MOS circuits. A generalization of the Weinberger array, gate matrix has been found to be particularly good for static CMOS technology. Regularity of structure and high density of devices are the salient features of this style, which is well suited for implementing complex gates, functional cells and random logic. A gate matrix consists of intersecting rows and columns to provide transistor placement and interconnections. The columns are realized in polysilicon and serve both as transistor gates and as interconnections. The rows are implemented with metal and diffusion. The diffusion layer forms a transistor at each intersection with a polysilicon gate and the metal lines are used for interconnections. Metal and diffusion can also be used to interconnect different rows. A second metal layer is available solely for the power and ground connections. Each row in the layout is called a *track*, and consists of one or more nets of the circuit.

An instance of the Gate Matrix Layout problem (henceforth GML) consists of a set of *nets* and their respective connections to a set of *gates*. A gate matrix layout consists of any assignment of tracks to the nets subject to the restriction that no two nets incident on a common gate can share the same track. The goal is to minimize the number of tracks utilized thereby minimizing the area required for a layout.

Figure 1 shows a CMOS circuit and two different layouts in gate matrix style for the circuit. The circuit consists of nets N_1, N_2 and N_3, and gates A, B, C, D, E, F and Z. Net N_1 is connected to gates A, B and F; net N_2 is connected to gates C, D and E; net N_3 connects gates C, F and Z.

A net consists of one or more transistors connected together either in series or in parallel using metal and diffusion. Since each net is typically connected only to a small subset of the gates, assigning each net to a distinct track may waste chip area. If several nets can share the same track, the total area of the layout may be reduced. Observe that the number of nets that can share a track depends on the linear ordering of the gates and the assignment of the nets to tracks. Therefore, by permuting the gates we may be able to obtain further savings in the number of tracks. For example, in Figure 1, the gate sequence $A-B-C-D-E-F-Z$ results in a layout requiring 3 tracks. On the other hand, the gate sequence $A-B-F-Z-C-D-E$ allows nets N_1 and N_2 to share a track and results in a layout requiring only 2 tracks.

Given an instance of GML, we wish to know if there is a permutation of the gates that will allow the circuit to be laid out in k or fewer tracks, where k is an input parameter. In the corresponding fixed-parameter problem, denoted GML(k), the number of tracks k is a fixed constant and not part of the input. The problem is to obtain a layout in no more than k tracks if possible.

Surprisingly, the combinatorial problem at the heart of Gate Matrix Layout arises in many different guises, even in fields as far removed from circuit layout as natural language processing [KoT].

2.2 Previous Work

Since its introduction, the gate matrix layout style has grown in popularity. Owing to their regular structure, it was expected that gate matrix layouts would be easy to generate using computers. However, since GML was proven to be $\mathcal{NP}$-Hard [KF], the emphasis has been on finding near-optimal solutions. GML has been studied

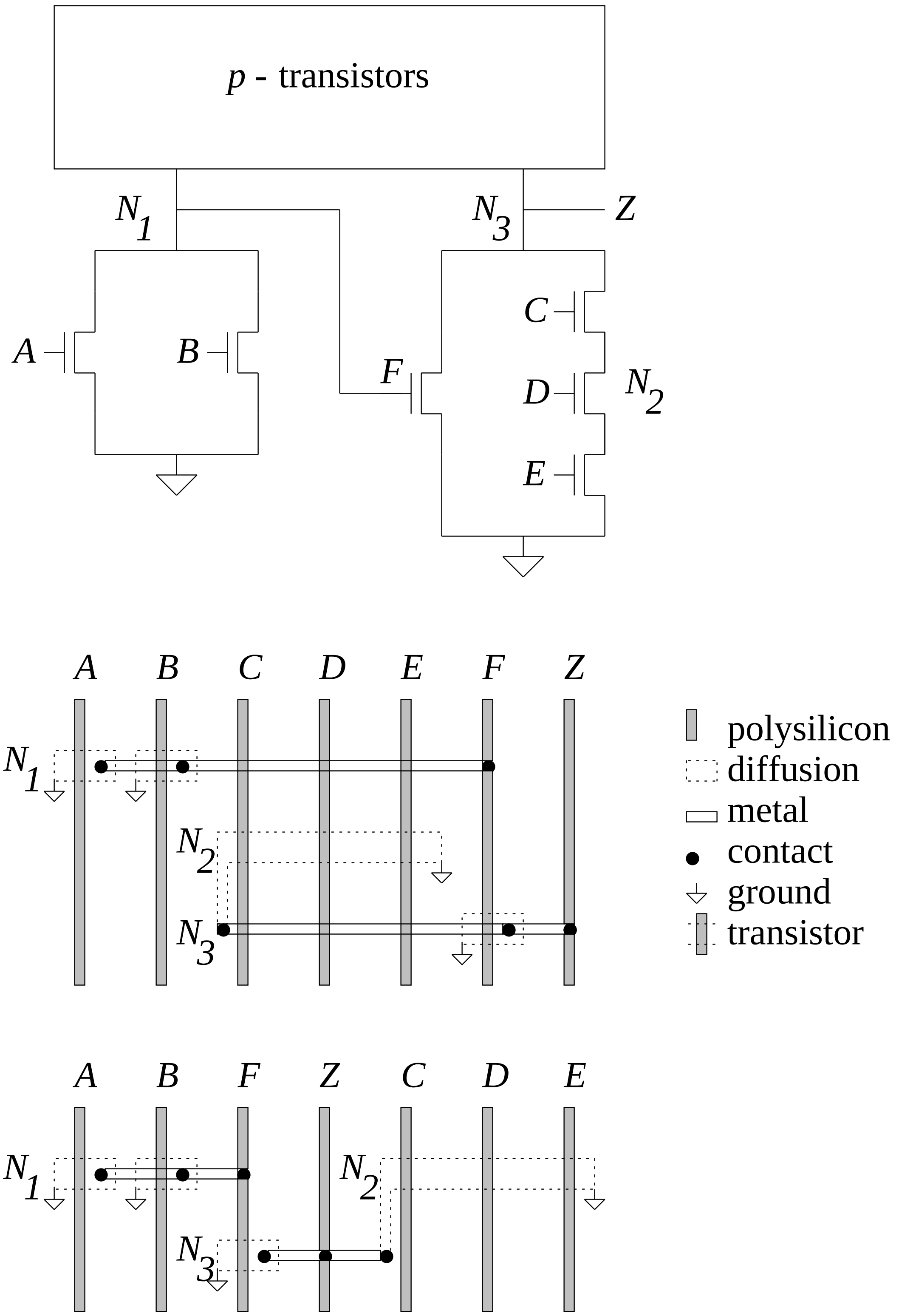

Figure 1: A CMOS circuit and its gate matrix layout

by numerous researchers ([DN, Li, WHW, Wi] for example), employing a variety of techniques. Most of these algorithms are heuristic, based on greedy strategies or branch and bound techniques. It has been shown [DKL] that such heuristics can produce arbitrarily bad results, and that unless $\mathcal{P} = \mathcal{NP}$, there can be no polynomial-time algorithm that will always produce a layout that is within a constant additive factor of the optimal.

Dynamic programming has also been proposed for GML [DKL]. Although this is an exact method, it is prohibitively slow. The algorithm presented in [DKL] has time complexity $O(m^2 2^m)$, where m is the number of gates. Its space requirement also grows exponentially with m. Notice that this is still an improvement over an exhaustive search algorithm that would explicitly consider all $m!$ column permutations.

Unlike the general version of GML, the fixed-parameter version is tractable [FL1]. That is, for any fixed k, polynomial-time algorithms to solve GML(k) are known. Before discussing the fixed-parameter version further, we develop some requisite theoretical background.

3 Problem Transformation

Given a circuit consisting of n nets and m gates, we can represent it as a 0-1 *incidence matrix* M, where

$$M(i,j) = \begin{cases} 1 & \text{if net } i \text{ is incident on gate } j \\ 0 & \text{otherwise} \end{cases}$$

The incidence matrix for the circuit in Figure 1 would be:

	A	B	C	D	E	F	Z
N_1	1	1	0	0	0	1	0
N_2	0	0	1	1	1	0	0
N_3	1	0	1	0	0	1	1

A 0-1 matrix has the *consecutive 1's* property if there exists a permutation of its columns such that all the 1's in each row occur consecutively. For example, the incidence matrix shown previously has the consecutive 1's property, realized with the column permutation $A-B-F-Z-C-D-E$. We formalize a well-known relationship between an instance of GML and the consecutive 1's property of its incidence matrix.

Lemma 3.1 A circuit has a gate matrix layout in k tracks if and only if there exists a (possibly empty) set of 0's in the corresponding incidence matrix M that, when changed to 1's, give a matrix M' with the consecutive 1's property such that the maximum column sum in M' does not exceed k.

Proof

($\Rightarrow$): Suppose the circuit has a k-track layout. Then it is safe to assume that each net is incident on every gate between its left-most and right-most gates in such a layout. By first permuting the columns of M in accordance with the order of the gates in the layout and then, in the permuted matrix by treating every 0 between the leftmost and the rightmost 1's in a row as a 1, we see that M has the consecutive 1's property. Moreover, the maximum column sum of M is k.

($\Leftarrow$): Suppose we can find a set of 0's to change into 1's such that M has the consecutive 1's property, and the maximum column sum of M is k. Then it is easy to assign tracks to the nets by sorting the nets according to their left end-point and following a simple greedy rule such that the circuit has a layout in k tracks (see [HS]). ∎

If M has the consecutive 1's property, the number of tracks required for a layout equals the maximum column sum. We can test whether M has the consecutive 1's property and also find a column permutation in $O(mn)$ time [BL]. Once we have a column permutation, we can complete the track assignment as mentioned before.

If M does not have the consecutive 1's property, then GML is $\mathcal{NP}$-Hard. We must find a column permutation such that when every 0 between the left-most and right-most 1's in each row is changed to a 1, the maximum column sum is minimized.

In the fixed-parameter problem GML(k), instead of minimizing the maximum column sum, we try to find a column permutation (if any exist) such that the number of 1's in any column is at most k.

We can also model a circuit by means of a graph as follows. The *intersection graph* corresponding to an incidence matrix is a graph $G = (V, E)$ where $V = \{v_1, \ldots, v_n\}$ is a set of n vertices, one for each net, and E is a set of edges such that $(v_i, v_j) \in E$ if and only if nets i and j are incident on the same gate. In other words, the 1's in each column of a matrix form a clique in its intersection graph. For example, the intersection graph of the circuit in Figure 1 is simply K_3, the complete graph on 3 vertices.

Notice that the mapping from an incidence matrix to an intersection graph is many-to-one. It is known [FL1] that the incidence matrix can be expanded to a format with exactly two 1's per column, so that each column in the expanded matrix corresponds to an edge of the intersection graph. Thus we can reverse the process to construct an incidence matrix for a graph. Although the intersection graph is an abstraction of a circuit, it captures the information essential for us to find a layout if any exist.

4 The Pathwidth Metric and Graph Minors

The pathwidth metric was first defined in [RS1]. Let G be a connected graph. A sequence $X_1, \ldots, X_r$ of subsets of the vertex set of G is a *path-decomposition* of G if the following conditions are satisfied.

i. For every edge e of G, some X_i $(1 \leq i \leq r)$ contains both ends of e.

ii. For $1 \leq i \leq j \leq k \leq r$, $X_i \cap X_k \subseteq X_j$.

The *pathwidth* of G is the minimum value of $k \geq 0$ such that G has a path-decomposition $X_1, \ldots, X_r$ with $|X_i| \leq k+1$ $(1 \leq i \leq r)$. The following theorem, which relates GML to the pathwidth metric of a graph, is fundamental to our algorithms.

Theorem 4.1 [FL3] A circuit has a gate matrix layout in no more than k tracks if and only if the corresponding intersection graph has pathwidth at most $k-1$.

A graph H is a *minor* of a graph G, denoted $H \leq_m G$, if and only if a graph isomorphic to H can be obtained from a subgraph of G by contracting edges. If H is a minor of G, and H is not isomorphic to G, then we write $H <_m G$. A family $\mathcal{F}$ of graphs is *closed* in the minor order if, for every G in $\mathcal{F}$, if H is a minor of G then H is also in $\mathcal{F}$.

Theorem 4.2 [RS1] For any fixed k, the family of graphs with pathwidth at most k is minor-closed.

A graph G is a *minimal* element of set Q if and only if, for every $H <_m G$, H is not in Q. If $\mathcal{F}$ is a minor-closed family, then the *obstruction set* for $\mathcal{F}$, written $obs(\mathcal{F})$, is the set of all minimal graphs in the complement of $\mathcal{F}$. Therefore, if $\mathcal{F}$ is a minor-closed family, then $G \in \mathcal{F}$ if and only if $H \not\leq_m G$ for every $H \in obs(\mathcal{F})$.

Theorem 4.3 [RS4] If $\mathcal{F}$ is a minor-closed family of finite graphs, then $obs(\mathcal{F})$ is finite.

Theorem 4.4 [RS3] For every fixed graph H, there exists a polynomial-time algorithm that, when given an input graph G, decides whether H is a minor of G.

Thus there is a polynomial-time algorithm to decide membership for any minor-closed family. The algorithm proceeds as follows. Given a graph G whose membership in $\mathcal{F}$ is to be decided, we check to see if G contains as a minor an obstruction to $\mathcal{F}$. If G contains no obstruction, then G belongs to $\mathcal{F}$. Otherwise G does not belong to $\mathcal{F}$. Since there are only a finite number of obstructions, and there exists a polynomial-time algorithm to test for the containment of each, membership in $\mathcal{F}$ can be decided in polynomial time.

Although Theorems 4.3 and 4.4 are powerful, they cannot be used directly by the algorithm designer. For one thing, Theorem 4.3 is non-constructive in the following sense: it proves that the number of obstructions is finite without actually showing what these obstructions are, or even how they can be found. It is known that any proof of Theorem 4.3 must be inherently non-constructive [FRS]; moreover, there can be no general scheme to compute the obstructions to a given family [FL3]. Furthermore, even if all obstructions were somehow known, the general algorithm for minor testing possesses enormous constants of proportionality. If the family $\mathcal{F}$ excludes a planar graph, then the graphs in $\mathcal{F}$ have bounded treewidth [RS2], and the minor containment test can be done in $O(n)$ time [Bod]. Alas, this algorithm also has extremely large constants.

Getting back to GML, it is known that for every k, there is a tree obstruction to the family of graphs with pathwidth k [FL1]. Thus it follows that GML(k) can, in principle, be solved in $O(n)$ time, although the algorithm is not practical.

5 Three-track Gate Matrix Layout

GML(1) is trivially solved. GML(2) can be solved in $O(nm)$ time using [BL]. The first genuinely difficult case is GML(3).

5.1 The Six Smallest Obstructions

It is known that there are exactly 110 obstructions to GML(3) [KL]. The number of vertices in these obstructions ranges from 4 to 22. Consider the six smallest obstructions, which we call A, B, C, D, E and F (see Figure 2). Graph A is K_4. Graph B is known as the Hajós graph. We are able to use the structure and relative density of these six obstructions to design fast minor containment tests for them. If we can develop fast minor containment tests for all 110 obstructions, then we would also have a fast decision algorithm for GML(3). However, the larger obstructions are sparser.

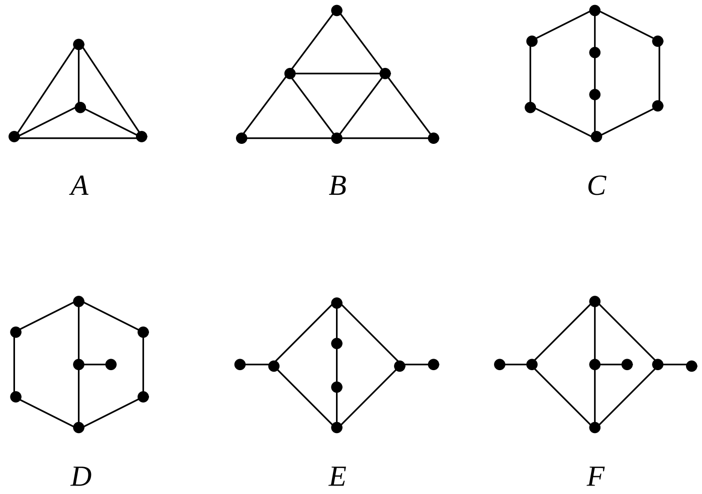

Figure 2: The six smallest obstructions to GML(3)

It appears to be very difficult to design tests for them.

Our algorithms for GML(3) are based on testing for the six obstructions alone. We have implemented these algorithms. Experimental results suggest that it is reasonable to expect that a circuit that cannot be laid out in three tracks will contain one of these six.

5.2 Testing for the Obstructions

Let G be a connected graph and let H be another graph. A *model of H in G* is a set $\{G_i : i \in V(H)\}$ of mutually disjoint, non-empty, connected subgraphs of G, together with a set $\{f_j : j \in E(H)\}$ of edges of G, such that $\forall j \in E(H)$ with endpoints i_1 and i_2, f_j has one endpoint in $V(G_{i_1})$ and one endpoint in $V(G_{i_2})$.

Observe that if there exists a model of H in G, then by contracting each G_i to a single vertex, we obtain a graph isomorphic to H. Moreover, every minor of G can be obtained this way.

Detecting Obstruction A

Given an intersection graph G corresponding to a circuit whose gate matrix layout is sought, we first check to see whether A is a minor of G. The following well known lemma is crucial to our algorithm.

Lemma 5.1 If G is a graph of order at least 4 and $K_4 \not\leq_m G$, then there exist a pair of non-adjacent vertices each with at most two neighbors in G.

In order to detect graph A, we use the following three graph-modifying rules. Given a vertex v, $N(v)$ denotes the set of neighbors of v.

```
The rules used for detecting obstruction A:
A1:  if (|N(v)| = 1) then delete vertex v;

Assumption:  Let {u, w} ⊆ N(v);
A2:  if (|N(v)| = 2 and (u, w) ∉ E) then
       begin
           add edge (u, w) to E;
           delete edge (u, v), edge (v, w) and vertex v;
       end;

A3:  if (|N(v)| = 2 and (u, w) ∈ E) then
           delete edge (u, v), edge (v, w) and vertex v;
```

We note that Rules A2 and A3 are very similar; we are not suggesting that they should necessarily be coded separately. Stating them this way, however, makes it easier to argue about their correctness later.

We repeatedly modify G using rules A1, A2 and A3, until we are left with a graph to which none of the rules apply. This algorithm is based on Duffin's [Du] characterization of series-parallel graphs as those that exclude graph A. A proof of correctness of these rules is contained in an appendix. If the resulting graph is empty, then we know that $K_4 \not\leq_m G$. Otherwise, the resulting graph has minimum degree at least three and according to Lemma 5.1, $K_4 \leq_m G$.

In the sequel, we assume that we have already checked and found that $K_4 \not\leq_m G$. This will allow us to take advantage of Lemma 5.1 and help simplify our algorithms for obstructions B, C, D, E and F.

Detecting Obstruction B

Our minor containment test for obstruction B is based on the following lemma.

Lemma 5.2 Obstruction $B \leq_m G$ if and only if there exist three disjoint, connected subgraphs G_1, G_2 and G_3 in G such that, contracting each G_i to a single vertex v_i results in a multigraph in which there are six vertex-disjoint paths, two between each pair v_i, v_j.

Proof

($\Rightarrow$): Let v_1, v_2 and v_3 be the three vertices of degree four in B. Label the other three vertices v_4, v_5 and v_6, so that v_4 is adjacent to v_1 and v_2, v_5 is adjacent to v_2 and v_3, and v_6 is adjacent to v_1 and v_3. If $B \leq_m G$, then there is a model of B in G consisting of non-empty connected subgraphs $G_{v_i}, \forall i, 1 \leq i \leq 6$. Thus the lemma may be satisfied by letting $G_1 = G_{v_1} \cup G_{v_4}$, $G_2 = G_{v_2} \cup G_{v_5}$, and $G_3 = G_{v_3} \cup G_{v_6}$.

($\Leftarrow$): If such G_1, G_2 and G_3 exist then, since G is a simple graph, there is a model of B in G. ∎

We refer to the three vertices v_1, v_2 and v_3 of this lemma as the *corners* of B.

In order to test whether B is a minor of a simple graph G, our algorithm uses an internal representation with weights on the edges (if the vertices of an edge-weighted graph are viewed as the contraction of mutually disjoint, connected subgraphs of a simple graph, then the weights denote the number of edges between these subgraphs). Initially, each edge of G is assigned weight 1. Since we only need two vertex-disjoint paths between any pair of corners, the weight of an edge is either 1 or 2.

We use the following three rules, defined for edge-weighted graphs, in order to detect B.

The rules used for detecting obstruction B:

B1: **if** $(|N(v)| = 1)$ **then** delete vertex v;

Assumption: Let $\{u, w\} \subseteq N(v)$ and weight$(u, v) \leq$ weight(v, w);

B2: **if** $(|N(v)| = 2$ and weight$(u, v) = 1$ and $(u, w) \notin E)$ **then**
    begin
        introduce edge (u, w);
        weight$(u, w) \leftarrow$ weight(v, w);
        delete edge (u, v), edge (v, w) and vertex v;
    end;

B3: **if** $(|N(v)| = 2$ and weight$(u, v) = 1$ and $(u, w) \in E)$ **then**
    begin
        weight$(u, w) \leftarrow 2$;
        delete edge (u, v), edge (v, w) and vertex v;
    end;

We repeatedly modify G using rules B1, B2 and B3 until no further modification is possible. If a vertex v has exactly two neighbors and both edges incident on v have weight greater than one, then we do not modify it. A proof of correctness of these rules can be found in an appendix.

5.3 Detecting Four Obstructions in One Pass

Testing for obstructions C, D, E and F is more complicated. We take advantage of the structural similarities among these four obstructions and devise a single algorithm that will detect the presence of any of them.

A *T-link* between two vertices u and v in a graph is a path of length two between u and v such that the intermediate vertex w on the path has a neighbor x distinct from u and v. All four vertices u, v, w and x, as well as the three edges $(u, w), (v, w)$ and (w, x) are considered to be part of the T-link. A *link* between two vertices u and v in a graph is either a path of length at least three between u and v, or a T-link between u and v.

Observe that each of the obstructions C, D, E and F has maximum degree three. Therefore, we can use the following lemma that relates the minor and topological orders. A graph H is topologically contained in a graph G (denoted $H \leq_t G$) if and only a graph isomorphic to H can be obtained by removing subdivisions and deleting vertices and edges from G (a subdivision is a vertex of degree two; we remove a subdivision by deleting the vertex and its incident edges, and adding an edge between its neighbors).

Lemma 5.3 [FL2] If H is a graph whose maximum degree does not exceed three, then $H \leq_m G$ if and only if $H \leq_t G$.

Our test for minor containment of C, D, E and F is based on the next lemma. The proof relies on the fact that in each of these obstructions, there exist only two vertices with exactly three neighbors such that each neighbor in turn has degree at least two. These two vertices will be called the *corners* of C, D, E or F. Note that there exist three mutually vertex-disjoint links between the corners of each obstruction.

Lemma 5.4 A graph G contains one of the obstructions C, D, E or F as a minor if and only if G contains two distinct vertices x and y and connected subgraphs G_1, G_2 and G_3 such that

i. $V(G_i) \cap V(G_j) = \{x, y\}$ for every $1 \leq i < j \leq 3$, and

ii. each G_i can be contracted to a link between x and y.

Proof ($\Rightarrow$): Suppose G contains such an obstruction as a minor. Then according to Lemma 5.3, G also contains the obstruction in the topological order. If $H \leq_t G$, then by subdividing zero or more edges of H we can obtain a graph $H' \subseteq G$. Hence, there exists a one-to-one mapping of the vertices of H into the vertices of G such that the edges of H map to vertex-disjoint paths in G. Let x and y be the corners of the obstruction. The subgraph consisting of the images of all the vertices and edges of each link between the corners is a connected subgraph of G. Since the images of the edges are vertex-disjoint paths, the three subgraphs corresponding to the three links intersect only at the corners.

($\Leftarrow$): Suppose there exist connected subgraphs G_1, G_2 and G_3 in G that satisfy conditions (i) and (ii) of the lemma. Then, depending on the type of link to which each subgraph can be contracted, G contains at least one of the obstructions C, D, E and F. ∎

For the purpose of testing whether any of the obstructions C, D, E and F is a minor of a simple graph G, our algorithm uses an internal representation with weights on the vertices as well as the edges of G. Recall that to test for obstruction B, we only used edge weights. Here, we also assign weights to the vertices in order to find T-links. The vertex and edge weights are initialized as follows.

The weight of a vertex is either 0 or 1. Initially, every vertex with three or more neighbors is assigned weight 1. All other vertices are assigned weight 0. The vertex weights are not changed during the algorithm.

The weight of an edge is an integer between 1 and 4. All edge weights are initialized to 1. A weight of 2 denotes a path of length two between the end-points. A weight of 3 means that we have found one link. A weight of 4 means that we have found two vertex-disjoint links between the end-points. The edge weights never decrease as the algorithm progresses.

We repeatedly use the following five rules, defined for graphs with both vertex and edge weights, in order to detect obstructions C, D, E and F. A proof of correctness of these rules is contained in an appendix.

The rules used for detecting obstructions C, D, E and F:

C1: **if** ($|N(v)| = 1$) **then** delete vertex v;

Assumption: Let $\{u, w\} \subseteq N(v)$ and weight$(u, v) \leq$ weight(v, w);

C2: **if** ($|N(v)| = 2$ and weight$(v, w) \leq 3$ and $(u, w) \notin E$) **then**
  begin
    add edge(u, w);
    weight$(u, w) \leftarrow$ minimum$\{3$, weight(u, v)+weight(v, w)+weight$(v)\}$;
    delete edge (u, v), edge (v, w) and vertex v;
  end;

C3: **if** ($|N(v)| = 2$ and weight$(v, w) \leq 3$ and weight$(u, w) < 3$) **then**
  begin
    weight$(u, w) \leftarrow$ minimum$\{3$, weight(u, v)+weight(v, w)+weight$(v)\}$;
    delete edge (u, v), edge (v, w) and vertex v;
  end;

C4: **if** ($|N(v)| = 2$ and weight$(v, w) \leq 3$ and weight$(u, w) = 3$ and
   weight(u, v)+weight(v, w)+weight$(v) \geq 3$) **then**
  begin
    weight$(u, w) \leftarrow 4$;
    delete edge (u, v), edge (v, w) and vertex v;
  end;

C5: **if** ($|N(v)| = 2$ and weight(u, v)+weight(v, w)+weight$(v) < 3$ and
   weight$(u, w) \geq 3$) **then** delete edge (u, v), edge (v, w) and vertex v;

5.4 A Linear Time Test for Obstructions

The algorithm that we use to test for minor containment of the graphs A, B, C, D, E and F is called MINOR-FREE. There are three distinct phases to this algorithm. The first phase is used to detect obstruction A, the second phase is used to detect B, and the third stage is used to detect the presence of any of C, D, E and F. The algorithm terminates as soon as an obstruction is detected. In each phase, MINOR-FREE initializes the vertex and edge weights of the graph and also the set of graph modification rules R appropriately and invokes the procedure REDUCE-GRAPH.

Algorithm REDUCE-GRAPH is the general scheme used to modify the input graph according to a set of specified rules. First a queue of all vertices that have only

Algorithm MINOR-FREE

Input : A connected simple graph G.
Output: YES, if none of A, B, C, D, E and F is a minor of G;
NO, if at least one of A, B, C, D, E and F is a minor of G.
begin procedure
1. Read the input graph G.
2. $R \leftarrow$ {A1,A2,A3};
   if (REDUCE-GRAPH(G, R) = NO) **then** output NO and **stop**;
3. Initialize the weight of each edge to 1;
   Initialize the weight of each vertex to 0;
4. $R \leftarrow$ {B1,B2,B3};
   if (REDUCE-GRAPH(G, R) = NO) **then** output NO and **stop**;
5. Initialize the vertex weights as follows:
   if ($|N(v)| \leq 2$) **then** weight(v) $\leftarrow 0$;
   else weight(v) $\leftarrow 1$;
6. R $\leftarrow$ {C1,C2,C3,C4,C5};
   if (REDUCE-GRAPH(G, R) = NO) **then** output NO;
   else output YES;

end procedure.

one or two neighbors is created. Then each vertex in the queue is visited sequentially. During a visit, the algorithm checks if any of the rules from the set R can be applied to that vertex. If a rule is applicable to the vertex, then edges are modified and the vertex is deleted. If this deletion causes any vertex to have exactly two neighbors then that vertex is added to the queue. The order of the graph decreases with each application of a rule. If none of the rules can be applied then the graph is either empty or contains an obstruction. The output of REDUCE-GRAPH is YES if the graph is empty, and NO otherwise. Therefore, Algorithm MINOR-FREE will always terminate with the correct result.

Lemma 5.5 Algorithm MINOR-FREE has a running time of $O(n)$.

Proof All operations performed by algorithm MINOR-FREE take constant time except for the three calls to algorithm REDUCE-GRAPH. The basic operations performed by algorithm REDUCE-GRAPH are: searching for vertices of degree three or

Algorithm REDUCE-GRAPH

Input : A connected graph G and a set R of rules.
Output: YES, if G can be reduced to the empty graph using R;
NO, otherwise.

```
begin procedure
1. Create a graph H isomorphic to G.
2. Create a queue of vertices in H with only one or two neighbors.
3. while (the queue is not empty) do
   begin
        remove a vertex v from the queue;
        if (a rule in R applies to v) then
        begin
             L ← list of neighbors of v with exactly three neighbors;
             modify H according to the rule;
             if (a vertex in L has degree two) then enqueue it;
        end;
        if (no rule in R applies to v) then mark v as visited;
   end;
4. if (H is empty) then output YES;
   else output NO;
end procedure.
```

less, deleting such vertices and the edges incident on them, checking for the existence of specific edges, and adding edges. Each of these operations can be performed in constant time using an uninitialized adjacency matrix representation for the graph. Since $K_4 \not\leq_m G \Rightarrow e \leq 2n-3$ [Bol], each operation is used $O(n)$ times, guaranteeing that REDUCE-GRAPH, and hence MINOR-FREE, runs in linear time. ∎

6 Layout Algorithms

Testing for the presence of obstructions is sufficient only to tell us whether or not a layout is possible. How can we find a layout when we know that it exists? We address this question here.

6.1 A Self-Reduction Technique

The term *self-reduction* is used to denote a process that solves the search version of a problem by making repeated calls to an algorithm for the decision version of that problem. Our implementation of the self-reduction algorithm from [BFL] uses the procedure FILL-IN, which follows. In attempting to change the at most nm 0's to 1's, FILL-IN makes $O(nm)$ calls to the decision algorithm. When no more changes are possible, a satisfactory column permutation, if one exists, can be found in $O(nm)$ time [BL].

We now list the entire layout procedure, which we call GATE-MATRIX-LAYOUT. If our decision algorithm is correct, so is our layout algorithm. But of course our decision algorithm is faulty. We shall explore this issue in the next section.

6.2 Imperfect Self-Reductions

Our fast decision algorithm, MINOR-FREE, tests for only six of the 110 obstructions to GML(3). If the input graph contains an obstruction other than these six, it will not be detected, giving us a "false positive." Alternately, even if the original graph has a three-track layout, we may introduce an obstruction during self-reduction, giving us a "false negative." Observe that false positives are inevitable. However, the occurrence of false negatives depends on the self-reduction scheme used and should be minimized.

One way to avoid false negatives entirely is to use a form of self-reduction in which we change a 0 to a 1 only if it does not amount to introducing a new edge in the graph. We call this algorithm SR1. SR1 never calls the decision algorithm MINOR-FREE and is a rather degenerate form of self-reduction. That is, SR1 only checks whether any incidence matrix of the graph has the consecutive 1's property. SR1 takes $O(nm)$ time. The most straightforward approach, which we call SR2, employs MINOR-FREE as its decision algorithm. SR2 takes $O(n^2m)$ time. A more expensive approach is to test whether the incidence matrix has the consecutive 1's property

Algorithm FILL-IN

Input : An $n \times m$ Boolean matrix M and
the corresponding intersection graph G.
Output: M after its entries have been filled in
for k-track gate matrix layout.

```
begin procedure
for j = 1 to m do
begin
    colsum ← number of 1's in column j of M
    i ← 1;
    while ((colsum ≤ k) and (i ≤ n)) do
    begin
        if (M(i,j) = 0) then
        begin
            M(i,j) ← 1;
            Add edges to G if necessary and remember them;
            Call the decision algorithm on G;
            if a layout is possible then colsum ← colsum + 1;
            else
            begin
                M(i,j) ← 0;
                Delete the edges that were added in G;
            end;
        end;
        i ← i + 1;
    end;
end;
end procedure.
```

after each change. This algorithm, SR3, takes $O(n^2m + nm^2)$ time.

How well do these imperfect self-reductions perform in practice? We explore this question in the next section.

7 Experimental Results

In order to study the behavior of these techniques, we implemented algorithm MINOR-FREE and the three self-reduction schemes SR1, SR2 and SR3. We also

Algorithm GATE-MATRIX-LAYOUT

Input : The netlist of a circuit consisting of n nets and m gates.
Output: A k-track gate matrix layout for the circuit, or
NO, if a k-track layout could not be found.

begin procedure
1. Read the netlist of the circuit.
2. Obtain the Boolean matrix M representing the circuit.
   Transform the matrix M to the intersection graph G.
3. Call the decision algorithm on G.
   if a layout is not possible **then**
   output NO and **stop**;
4. Call Algorithm FILL-IN on M and G.
5. Test whether M has the consecutive 1's property.
   if M does not have the consecutive 1's property **then**
   output NO and **stop**;
   else find a good column permutation of M;
6. Sort the n nets into m buckets according to their left end-point.
   Assign a track to each net using a greedy strategy [HS].
7. Output the layout.

end procedure.

implemented the algorithm of [FG] to determine whether a 0-1 matrix has the consecutive 1's property (although this algorithm takes $O(n^2m)$ time in contrast to the $O(nm)$ time of [BL], we chose it for ease of implementation). These programs were written in the C language and were tested on a Sun SPARC-20 workstation. We compare our results with those produced by the exact dynamic programming formulation of [DKL].

7.1 Randomly Generated Inputs

We tested our algorithm on large numbers of randomly generated inputs. For each fixed value of n and probability p, we generated a hundred random graphs of order n with uniform edge probability p and treated each graph as if it were the intersection graph of a circuit with n nets. The incidence matrix corresponding to such a random graph was obtained by creating a row (net) for each vertex in the graph, a column (gate) for each edge, and putting two 1's in each column in the positions corresponding to the end-points of the edge and 0's everywhere else.

7.2 Tabular Data

Tables 1 and 2 show some representative results from our experiments. In each table, the graphs tested are grouped by edge-probability, whose value is shown in the first column. The second column shows the number of vertices in the graph. Since we test for all obstructions of order less than 8, ours is an exact algorithm for graphs with 7 or fewer vertices. Therefore, we start our experiments with graphs of order 8. For any given edge-probability, large graphs have a greater chance of containing an obstruction than small graphs; beyond a certain size, this probability is so overwhelming that we find no candidates for self-reduction. We terminate our experiments at this point.

Columns three through five show results obtained using [DKL]. The third column lists the number of layouts found. The fourth column lists the number of inputs for which no layout is possible. The fifth column lists the number of inputs [DKL] could not resolve (each input was given at least four hours of CPU time before it was halted). The sixth column shows the result of using algorithm MINOR-FREE to decide whether the circuit has a three-track layout. The value in this column is never less than the actual number of layouts that exist; any difference is due to the occurrence of false positives.

Columns seven, eight and nine show the results of self-reduction. Each circuit that satisfies MINOR-FREE is self-reduced using SR1, SR2 and SR3. Not surprisingly, SR1 is the worst of the three, because the incidence matrices of a large number of the graphs generated do not have the consecutive 1's property. SR2 is a tremendous improvement over SR1 and in fact is fairly close to the exact algorithm for small graphs. SR3 turns out to be only a marginal improvement over SR2, and probably does not justify its additional computing time.

As the graphs tested grow larger and sparser, we tend to encounter more false negatives, since we are apt to introduce a (large) obstruction during self-reduction. Even so, our algorithm finds a layout almost as often as [DKL], and does so in drastically far less time. To illustrate, with the edge-probability set at 0.15 and the number of vertices set at 14, [DKL] takes an average of 4.2 seconds per graph to find a layout; in contrast, SR2 takes 0.018 seconds. More dramatically, with the edge-probability set at 0.075 and the number of vertices set at 25, and (conservatively) counting only four hours of CPU time for the input instances on which [DKL] does not terminate, the numbers are 91 minutes and 0.09 seconds, respectively.

The DKL code we employed has already been highly optimized for speed, for example by pruning pendant paths in the input graph and by collapsing edges that belong to a clique [Ki]. Therefore, it is unlikely that its performance can be improved significantly. On the other hand, the code for our algorithm was written primarily as a proof of concept without much effort at optimization. It should be possible to enhance its performance substantially. Furthermore, it strikes us as likely that false negatives can be reduced with some more complicated self-reduction scheme.

Table 1: Experimental results on small graphs

edge probability	no. of vertices	dynamic programming: yes	no	halted	MINOR-FREE: layout possible	self-reduction: layout found SR1	SR2	SR3
0.375	8	49	51	0	49	23	49	49
	9	20	80	0	20	2	19	19
	10	6	94	0	7	3	6	6
	11	1	99	0	1	0	1	1
	12	0	100	0	0	0	0	0
0.25	8	88	12	0	88	51	88	88
	9	82	18	0	82	35	79	80
	10	62	38	0	63	22	62	62
	11	45	55	0	48	7	42	42
	12	22	78	0	24	6	19	20
	13	12	88	0	12	2	8	9
	14	3	97	0	4	0	3	3
	15	1	99	0	1	0	1	1
	16	0	100	0	0	0	0	0
0.15	8	100	0	0	100	85	100	100
	9	100	0	0	100	83	100	100
	10	98	2	0	98	66	97	97
	11	97	3	0	98	54	95	95
	12	89	11	0	89	38	82	83
	13	82	18	0	82	20	72	73
	14	65	35	0	69	21	61	61
	15	51	49	0	53	9	33	33
	16	34	66	0	37	10	24	22
	17	26	49	25	33	2	14	14
	18	16	23	61	25	2	14	14
	19	7	14	79	13	0	3	3
	20	2	3	95	10	0	2	2
	21	0	1	99	1	0	1	1
	22	0	0	100	1	0	0	0
	23	0	1	99	1	0	0	0
	24	0	0	100	0	0	0	0

Table 2: Experimental results on larger graphs

edge probability	no. of vertices	dynamic programming			MINOR-FREE: layout possible	self-reduction: layout found		
		yes	no	halted		SR1	SR2	SR3
0.1	15	98	2	0	99	55	82	82
	20	40	20	40	56	9	32	32
	25	1	0	99	15	1	4	4
	30	0	0	100	0	0	0	0
0.075	20	79	12	9	85	28	67	67
	25	27	2	71	66	7	22	23
	30	5	0	95	25	2	9	9
	35	0	0	100	3	0	0	0
	40	0	0	100	0	0	0	0
0.05	20	100	0	0	100	74	94	94
	25	86	1	13	98	48	79	79
	30	43	4	53	84	22	42	42
	35	14	2	84	64	3	13	13
	40	2	0	98	30	0	1	1
	45	0	0	100	5	0	0	0
	50	0	0	100	2	0	0	0
	55	0	0	100	0	0	0	0
0.0375	25	98	0	2	100	73	95	95
	30	87	0	13	100	54	78	78
	35	57	4	39	88	23	42	42
	40	29	0	71	78	10	18	18
	45	11	0	89	49	2	7	7
	50	1	0	99	33	0	2	2
	55	0	0	100	13	0	0	0
	60	0	0	100	4	0	0	0
	65	0	0	100	0	0	0	0
0.025	30	100	0	0	100	94	98	98
	40	89	0	11	100	55	78	78
	50	38	0	62	92	21	41	41
	60	12	0	88	66	4	15	15
	70	1	0	99	30	1	1	1
	80	0	0	100	5	0	0	0
	90	0	0	100	0	0	0	0

8 Summary

Our main intent has been to illustrate the utility of an obstruction-based approach to algorithm design. Many well-known graph families are amenable to this method [FL4]. Because the number of obstructions to such families is generally a steep function of a relevant input parameter, our technique may merit further study whenever (i) a small set of key obstructions can be identified, and (ii) a fast test for them can be devised.

References

[BFL] D. J. Brown, M. R. Fellows, and M. A. Langston, "Polynomial-Time self-reducibility: theoretical motivations and practical results," *International Journal of Computer Mathematics* 31 (1989), 1–9.

[BL] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms," *Journal of Computer and Systems Science* 13 (1976), 335–379.

[Bol] B. Bollobás, "Extremal Graph Theory," Academic Press, New York (1978).

[Bod] H. L. Bodlaender, "A linear time algorithm for finding tree-decompositions of small treewidth," *Proceedings, 25th Annual ACM Symposium on Theory of Computing* (1993), 226–234.

[DKL] N. Deo, M. S. Krishnamoorthy, and M. A. Langston, "Exact and approximate solutions for the Gate Matrix Layout problem," *IEEE Transactions on Computer-Aided Design* 6 (1987), 79–84.

[DN] S. Devadas and R. Newton, "Topological optimization of multiple-level array logic," *IEEE Transactions on Computer-Aided Design* 6 (1987), 915–940.

[Du] R. J. Duffin, "Topology of series-parallel networks," *Journal of Mathematical Analysis and Applications* 10 (1965), 303–318.

[FG] D. R. Fulkerson and O. A. Gross, "Incidence matrices and interval graphs," *Pacific Journal of Mathematics*, 15:3 (1965), 835–855.

[FL1] M. R. Fellows and M. A. Langston, "Nonconstructive advances in polynomial-time complexity," *Information Processing Letters*, 26 (1987/88), 157–162.

[FL2] M. R. Fellows and M. A. Langston, "Nonconstructive tools for proving polynomial-time decidability," *Journal of the ACM*, 35:3 (1988), 727–739.

[FL3] M. R. Fellows and M. A. Langston, "On search, decision and the efficiency of polynomial-time algorithms," *Proceedings, 21st Annual ACM Symposium on Theory of Computing* (1989), 501–512.

[FL4] M. R. Fellows and M. A. Langston, "On well-partial-order theory and its application to combinatorial problems of VLSI design," *SIAM Journal on Discrete Mathematics* 5:1 (1992), 117–126.

[FRS] H. Friedman, N. Robertson, and P. Seymour, "The metamathematics of the graph minor theorem," *Contemporary Mathematics* 65 (1987), 229–261.

[HS] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," *Proceedings, 8th Design Automation Workshop* (1971), 155–169.

[KF] T. Kashiwabara and T. Fujisawa, "NP-Completeness of the problem of finding a minimum-clique-number interval graph containing a given graph as a subgraph," *Proceedings, International Symposium on Circuits and Systems* (1979), 657–660.

[Ki] N. G. Kinnersley, Private Communication.

[KL] N. G. Kinnersley and M. A. Langston, "Obstruction set isolation for the Gate Matrix Layout problem," *Discrete Applied Mathematics* 54 (1994), 169–213.

[KoT] A. Kornai and Z. Tuza, "Narrowness, pathwidth, and their application in natural language processing," *Discrete Applied Mathematics* 36 (1992), 87–92.

[La] M. A. Langston, "An Obstruction-Based Approach to Layout Optimization," Graph Structure Theory, (N. Robertson and P. D. Seymour, editors), AMS Press, 1993, 623–630.

[Le] T. Lengauer, "Combinatorial algorithms for integrated circuit layout," John Wiley & Sons, New York, 1990.

[Li] J-T. Li, "Algorithms for Gate Matrix Layout", *Proceedings, International Symposium on Circuits and Systems* (1983), 1013–1016.

[LL] A. D. Lopez and H-F. S. Law, "A dense gate matrix layout method for MOS VLSI," *IEEE Transactions on Electron Devices* ED-27 (1980), 1671–1675.

[RS1] N. Robertson and P. D. Seymour, "Graph Minors I. Excluding a forest," *Journal of Combinatorial Theory, Series B* 35 (1983), 39–61.

[RS2] N. Robertson and P. D. Seymour, "Graph Minors V. Excluding a planar graph," *Journal of Combinatorial Theory, Series B* 41(1986), 92–114.

[RS3] N. Robertson and P. D. Seymour, "Graph Minors XIII. The disjoint paths problem," manuscript (1986).

[RS4] N. Robertson and P. D. Seymour, "Graph Minors XX. Wagner's Conjecture," manuscript (1988).

[WHW] O. Wing, S. Huang, R. Wang, "Gate Matrix Layout," *IEEE Transactions on Computer-Aided Design* 4 (1985), 220–231 .

[Wi] O. Wing, "Interval-graph-based circuit layout," *Proceedings, International Conference on Computer-Aided Design* (1983), 84–85.

APPENDICES

A Proof of Correctness of Rules A1, A2, A3

Lemma A.1 If G' is obtained from G by an application of rule A1, A2 or A3, then $K_4 \leq_m G'$ if and only if $K_4 \leq_m G$.

Proof Suppose G' is obtained from G by an application of rule A1, A2 or A3.
($\Rightarrow$): If G' is obtained from G by using rule A1 or A3, then $G' \subset G$. Rule A2 is equivalent to contracting v to u along the edge (u, v). Therefore, $G' \leq_m G$ in all three cases. Hence, $K_4 \leq_m G' \Rightarrow K_4 \leq_m G$.
($\Leftarrow$): Let $v \in V(G)$ and $\delta(v) \leq 2$. Suppose v belongs to the connected subgraph G_i in the model M of a K_4. Then a neighbor u of v and the edge (u, v) are also in G_i because $\delta(K_4) = 3$. Observe that each of the rules A1, A2 and A3 simply contracts the edge (u, v) to u. After this contraction, G_i becomes a connected graph G_i'. By replacing G_i with G_i' in M, we obtain a model M' of K_4 in G'. ∎

B Proof of Correctness of Rules B1, B2, B3

For the purpose of this proof, edge-weighted graphs will be distinguished with a hat (ˆ).

The *simplification* of an edge-weighted graph $\hat{G}$ is the simple graph G obtained as follows:

i. Let G be the simple graph obtained by ignoring the edge weights of $\hat{G}$.

ii. For each edge (u, w) with weight 2 in $\hat{G}$, introduce a new vertex v in G and make v adjacent to both u and w.

Note that every edge-weighted graph has a unique simplification. For example, the simplification of graph $\hat{B}$ shown in Figure 3 is obstruction B.

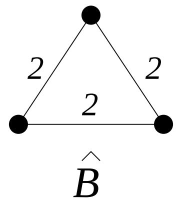

Figure 3: A graph with edge weights

Lemma B.1 If $\hat{G}'$ is obtained from $\hat{G}$ by an application of rule B1, B2 or B3, then B is a minor of the simplification of $\hat{G}'$ if and only if B is a minor of the simplification of $\hat{G}$.

Proof Suppose $\hat{G}'$ is obtained from $\hat{G}$ by an application of rule B1, B2 or B3. Let G and G' be the simplification of $\hat{G}$ and $\hat{G}'$ respectively.
($\Rightarrow$): Suppose $B \leq_m G'$. If rule B1 was used, then $\hat{G}'$ is the same as $\hat{G} - \{v\}$. Therefore, $G' \leq_m G$. Applying rule B2 is equivalent to contracting vertex v to u along edge (u, v) both in $\hat{G}$ and in G. Hence, $G' \leq_m G$. If rule B3 was used, then $\hat{G}' - (u, w) \subset \hat{G} - (u, w)$. Since weight$(u, w)$ in $\hat{G}'$ is 2, its simplification results in an edge (u, v) and a path of length two between u and v in G'. The intermediate vertex on this path is adjacent only to u and w. The edge (u, w) exists in G also. Moreover, since u and v are neighbors of v in G, the edges $(u, v), (v, w)$ constitute a path of length two between u and w. Since $v \notin G'$, $G' \leq_m G$.
($\Leftarrow$): Let $B \leq_m G$. Then G must satisfy Lemma 5.2. Consider the structure of the mutually disjoint, connected subgraphs G_i of that lemma. Since v has at most two neighbors in G, none of $V(G_1), V(G_2)$ and $V(G_3)$ consists solely of v. Assume that if $u \in G_i$, then $i = 1$.

If rule B1 is used to obtain $\hat{G}'$, then every path from v passes through u in G. Therefore, if $v \in G_i$, then $u \in G_i$ also. Then, the subgraphs $G_1 - \{v\}, G_2, G_3$ exist in G' and satisfy Lemma 5.2. Hence, $B \leq_m G'$.

Suppose rule B2 or B3 is used to obtain $\hat{G}'$. Then, deleting the edge (u, w) from $\hat{G}'$ results in the same graph as deleting vertex v and edge (u, w) from $\hat{G}$. The rest of the proof is based on this fact.

Since w and u are connected in G, we can assume that if $u \in G_1$, then w is in some G_i. If u and w are both in G_1, then the subgraphs G_2 and G_3 also exist in G'. In G', the subgraph induced by $V(G_1)$ can be taken instead of G_1. Since u and w are connected in G', $B \leq_m G'$.

If $u \in G_1$ and $w \in G_2$, then the two vertex-disjoint paths between G_1 and G_2 in G that are necessary to form obstruction B also exist between $G_1 - \{v\}$ and G_2 in G'. Moreover, these two paths are disjoint from the other four paths required by Lemma 5.2, both in G and G'. Therefore, $B \leq_m G'$.

If neither u nor w is in any G_i, then the subgraphs G_i also exist in G'. Moreover, at most one path between some G_i and G_j can include both u and w in G. Since the edge (u, w) exists in G', G' satisfies Lemma 5.2. Therefore, $B \leq_m G'$. ∎

We repeatedly modify $\hat{G}$ using rules B1, B2 and B3 until no further modification is possible. If a vertex v has exactly two neighbors and both edges incident on v have weight greater than one, then we do not modify it.

Lemma B.2 If none of the rules B1, B2 and B3 is applicable to $\hat{G}$, then B is a minor of the simplification of $\hat{G}$.

Proof The proof is by induction on $|V(\hat{G})|$. In order for none of B1, B2 and B3 to be applicable, $|V(\hat{G})| \geq 3$. Let G be the simplification of $\hat{G}$. If $|V(\hat{G})| = 3$, then each edge of $\hat{G}$ has weight 2 and $B \leq_m G$. Assume that the lemma holds whenever $3 \leq |V(\hat{G})| \leq n$.

Suppose $\hat{G}$ is a graph of order $n + 1$. Since rule B1 cannot be applied to $\hat{G}$, each vertex in $\hat{G}$ has at least two neighbors. Let v be a vertex with exactly two neighbors u and w in $\hat{G}$ (v exists because $K_4 \not\leq_m G$). Both edges incident on v have weight 2, because rules B2 and B3 cannot be applied. If $(u, w) \notin E$, then contracting v to u along edge (u, v) results in a graph $\hat{G}'$ to which none of the rules can be applied. Let

G' be the simplification of $\hat{G}'$. Since $|V(\hat{G}')| = n$, it follows that $B \leq_m G'$. Further, since $G' \leq_m G$, we have $B \leq_m G$.

If the edge (u, w) exists and weight$(u, w) \geq 2$, then the graph $\hat{B}$ (see Figure 3) is a subgraph of $\hat{G}$. Hence, $B \leq_m G$. The only remaining possibility is that weight(u, w) $= 1$.

We know that any graph has a biconnected component that contains at most one of the articulation points in the graph. Let $\hat{H}$ be such a biconnected component of $\hat{G}$ and let H be the simplification of $\hat{H}$. We will show that $B \leq_m H$.

We know that $K_4 \not\leq_m H$, because $K_4 \not\leq_m G$ and $H \subseteq G$. According to Lemma 5.1, there exists a vertex $v \in \hat{H}$, such that v has exactly two neighbors in $\hat{H}$ and v is not an articulation point of $\hat{G}$. Let $N(v) = \{u, w\}$. Then weight$(u, v) =$ weight(v, w) $= 2$, and weight$(u, w) = 1$. One of u and w, say w, is not an articulation point of $\hat{G}$. Since rules B2 and B3 are not applicable, w has a third neighbor x (besides u and v) in $\hat{H}$. Since $\hat{H}$ is biconnected, there exist vertex-disjoint paths connecting u and w to x. These two paths and the edge (u, w) together make two vertex-disjoint paths between u and w that are disjoint from v in $\hat{H}$. Therefore, according to Lemma 5.2, $B \leq_m H$. Since $H \subseteq G$, we conclude that $B \leq_m G$. ∎

C Proof of Correctness of Rules C1–C5

Graphs with both vertex and edge weights will be distinguished with a check (ˇ).

An *expansion* of a weighted graph $\check{G}$ is a simple graph G obtained using the following sequence of five steps:

i. Let G be the simple graph obtained by ignoring the weights of $\check{G}$.

ii. If an edge of $\check{G}$ has weight 2, then subdivide the corresponding edge in G.

iii. If an edge has weight 3 in $\check{G}$, then replace that edge in G with a link of any kind.

iv. If an edge has weight 4 in $\check{G}$, replace that edge in G with a pair of vertex-disjoint links of any kind.

v. Finally, if a vertex v of weight 1 in $\check{G}$ has fewer than three neighbors in G, then add a new vertex to G and make it adjacent to v.

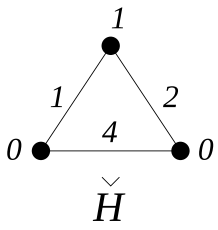

Figure 4: A graph with vertex and edge weights

In general, a graph $\check{G}$ can have more than one expansion. For example, each of the obstructions C, D, E and F is a minor of an expansion of graph $\check{H}$ shown in Figure 4. The expansion of a graph is unique only if its maximum edge weight is 2.

Lemma C.1 If $\check{G}'$ is obtained from $\check{G}$ by an application of rule C1, C2, C3, C4 or C5, then an expansion of $\check{G}'$ contains one of the obstructions C, D, E and F as minor if and only if an expansion of $\check{G}$ does.

Proof

($\Rightarrow$): Suppose G' is an expansion of $\check{G}'$ that contains one of C, D, E and F as a minor. If $\check{G}'$ was obtained using rule C1 or C5, then $\check{G}'$ is the same as $\check{G} - \{v\}$. Therefore, G' is a minor of an expansion of $\check{G}$.

If one of the rules C2, C3 and C4 was used, then deleting the edge (u, w) from $\check{G}'$ results in a graph identical to that obtained by deleting vertex v and edge (u, w) from $\check{G}$. If rule C2 or C3 was used, then weight(u, w) in $\check{G}' \leq$ weight(u, v) + weight(v, w) + weight(v) in $\check{G}$. Therefore, G' is a minor of an expansion of $\check{G}$.

Let $\check{H}'$ be the subgraph induced by u and w in $\check{G}'$, and let $\check{H}$ be the subgraph induced by u, v and w in $\check{G}$. Consider the expansion $H' \subseteq G'$ of $\check{H}'$ and an expansion

H of $\check{H}$.

If rule C4 was used, then weight(u, w) = 4 in $\check{G}'$. Hence, there are two vertex-disjoint links between u and w in H'. Since weight(u, w) = 3 in $\check{G}$, there exists a link in H between u and w that excludes vertex v. Further, since weight(u, v)+weight(v, w)+weight$(v) \geq 3$, a link between u and w in H that includes vertex v can be obtained by contracting zero or more edges. Therefore, G' is a minor of an expansion of $\check{G}$.

($\Leftarrow$): Let G be an expansion of $\check{G}$ that contains one of the four obstructions. We show that an expansion G' of $\check{G}'$ also contains an obstruction.

If an expansion of $\check{G} - \{v\}$ contains an obstruction, then so does some expansion of $\check{G}'$. This is because for rules C1 and C5 $\check{G}'$ is identical to $\check{G} - \{v\}$. For rule C2, the edge (u, w) does not exist $\check{G}$. For rules C3 and C4, the weight of edge (u, w) in $\check{G}'$ is not less than its weight in $\check{G}$. Therefore, every expansion of $\check{G} - \{v\}$ is a subgraph of an expansion of $\check{G}'$.

Suppose no expansion of $\check{G} - \{v\}$ contains an obstruction. Let G_1, G_2 and G_3 be the connected subgraphs, and let x and y be the corners of an obstruction in G that satisfy Lemma 5.4. Notice that only those vertices that were originally in $\check{G}$ and none of the vertices introduced during expansion can serve as corners in G. Since v has at most two neighbors in $\check{G}$, it cannot be a corner in G . Thus v must be an essential part of a link between the corners of any obstruction in G. Therefore, assume that v is in the connected subgraph G_1 of G.

If v is part of a T-link between the corners, then it must be the intermediate vertex of the T-link. Otherwise, the intermediate vertex of the T-link would have weight 1 in both $\check{G}$ and $\check{G} - \{v\}$ and as a result, an expansion of $\check{G} - \{v\}$ would contain the same obstruction as G. Therefore, if v has only one neighbor in $\check{G}$, then it cannot be part of a link between the corners. Hence, v must have at least two neighbors in $\check{G}$, and rule C1 is not applicable. Let $N(v) = \{u, w\}$.

Suppose v is part of a T-link between the corners. Then v is the intermediate vertex of T-link between u and w, and weight$(v) = 1$. Therefore, u and w must also be in G_1 besides v. Also, weight(u, v) + weight(v) + weight$(v, w) \geq 3$ in $\check{G}$.

If v does not form a T-link, then it must be part of a path of length at least three between the corners. Therefore, u and w must also be in G_1 besides v. Observe that G_1 can be contracted to a path of length three between u and w that also includes v if and only if weight(u, v)+ weight$(v, w) \geq 3$ in $\check{G}$.

Suppose weight(u, v)+ weight(v)+ weight$(v, w) \geq 3$. In this case, rule C5 cannot be applied. Let $\check{G}'$ be obtained by applying one of the rules C2, C3 and C4. Since the number of vertex-disjoint links between u and w in any expansion G' of $\check{G}'$ is the same as the number in G, we conclude that G' also contains an obstruction.

Suppose weight(u, v)+ weight(v)+ weight$(v, w) < 3$. In this case, rule C4 cannot be applied. Since v has to be part of a path of length at least three between the corners, u and w cannot both be corners. Hence, if the edge (u, w) exists, then weight$(u, w) = 1$. Therefore, rule C5 cannot be applied. Let $\check{G}'$ be obtained by applying rule C2 or C3. Then the length of the longest path between u and w in G' is the same as that in G. Therefore, G' contains an obstruction also. This completes the proof. ∎

Lemma C.2 If none of the rules C1, C2, C3, C4 and C5 can be applied to $\check{G}$, then an expansion of $\check{G}$ contains one of the obstructions C, D, E and F as a minor.

Proof Since rule C1 cannot be applied, each vertex of $\check{G}$ has at least two neighbors. Since K_4 is not a minor of any expansion of $\check{G}$, there exists a vertex v with exactly two neighbors in $\check{G}$. Let $N(v) = \{u, w\}$. As in the rules C2–C5, assume that weight$(u, v) \leq$ weight(v, w). There are two cases to consider.

Case 1: weight$(u, v) \leq$ weight$(v, w) \leq 3$.

In order for v not to meet the conditions of rules C2–C5, the edge $(u, w) \in E$, weight$(u,w) > 3$, and weight(u,v) + weight(v,w) + weight$(v) \geq 3$. Consider the subgraph $\check{H}$ induced by vertices u, v and w in $\check{G}$. There exist two links between u and w, and the vertex v along with its two edges is the third link. Therefore, $\check{H}$ can be expanded into an obstruction.

Case 2: weight$(u,v) \leq$ weight$(v,w) = 4$.

The proof of this is by induction on $|V(\check{G})|$. If $|V(\check{G})| = 3$, then every vertex of $\check{G}$ has two neighbors. If weight$(u,v) \leq 3$ and weight$(u,w) \leq 3$, then this reduces to Case 1. Otherwise, at least one of the edges (u,v) and (u,w) has weight 4. As a result, an expansion of $\check{G}$ contains an obstruction.

Suppose the lemma is true for all $3 \leq |V(\check{G})| \leq n$. Let $|V(\check{G})| = n+1$, and let $v \in V(\check{G})$ with $N(v) = \{u,w\}$.

If weight$(u,v) < 3$, then we contract the edge (u,v) to obtain $\check{G}'$. Since $|V(\check{G}')| = n$, an expansion G' of $\check{G}'$ contains an obstruction. Moreover, there exists an expansion G of $\check{G}$ such that $G' \leq_m G$. Therefore, G also contains the obstruction.

If weight$(u,v) \geq 3$, let $\check{H}$ be a biconnected component of $\check{G}$ containing only one articulation point of $\check{G}$. There exists a vertex $v \in V(\check{H})$ such that $N(v) = \{u,v\}$ and v is not an articulation point of $\check{G}$. Moreover, there exists a path from u to w in $\check{H}$ that does not include v. Since weight$(v,w) = 4$ and weight$(u,v) \geq 3$, the subgraph of $\check{H}$ induced by u, v and w, along with this path between u and w can be expanded to a simple graph that contains an obstruction. Therefore, an expansion of $\check{G}$ also contains the obstruction. ∎

D Software Package (SIXPACK)

Our software package (which we humorously dub SIXPACK) is written entirely in C. It is portable to any platform with an ANSI C compiler. Although SIXPACK can be used in stand-alone fashion, any of the constituent functions of the package can easily be used independently with the appropriate data structures. This is facilitated by the user interface, which permits either interactive use or invocation from within another program.

The package also contains routines supplementary to the main aim of obstruction testing and self-reduction. Included is an implementation of the algorithm, due to Fulkerson and Gross, that tests if an input matrix has the consecutive 1's property. This routine may be used to find a three-track layout after a successful self-reduction. Also included are routines that generate random graphs of specified order and edge density and random matrices of specified size and number of non-zero entries. These are helpful in evaluating the effectiveness of our approach on graphs and matrices of varying size and sparseness.

Source code for SIXPACK, comprising our implementation of algorithm MINOR-FREE and three different versions of self-reduction, along with a makefile to help compile the programs, is available as a tar file. Our program accepts either a graph or a 0-1 matrix as input and can also generate its own input using random graph generation and random matrix generation routines. Various options to the program can be set so that a simple obstruction test or one of three variants of self-reduction is performed.

Since our algorithm is graph-based, input matrices are converted into their intersection graphs for processing. Graphs are stored in a simple adjacency list format. Each graph has a header node that contains information about the graph and a pointer to an array of vertex nodes. Each vertex node contains fields that store the label, degree and weight (a parameter used by algorithm MINOR-FREE) of the corresponding vertex and a pointer to a list of edge nodes. Each node in a list of edge nodes contains fields that store one endpoint and the weight of the corresponding edge, and a pointer to the next node in the list. We prefer the adjacency list representation over the adjacency matrix representation because input graphs that do not contain an obstruction must have fewer than twice as many edges as vertices. Since the adjacent list representation uses only linear space to store such graphs as opposed to the quadratic space used by the adjacency matrix representation, we expect input graphs of interest to have a comparatively compact representation in adjacency list form.

Storing graphs in adjacency list format also allows us to implement efficiently such basic operations on graphs as finding connected and biconnected components, which are used by algorithm MINOR-FREE. Besides these, the only modifications performed by MINOR-FREE on the input graph are local to vertices of degree two or less, and these can be performed quickly on the adjacency list representation as well. The algorithms for self-reduction require one more operation, the addition of a specified edge, which can be done in constant time with our representation.