A Practical Approach to Layout Optimization*

Rajeev Govindan, **Michael A. Langston** and
Siddharthan Ramachandramurthi

Department of Computer Science
University of Tennessee
Knoxville, TN 37996–1301
USA

October 9, 1992

Abstract

In this paper, we present a practical approach for finding optimal solutions for fixed-parameter versions of many VLSI layout problems. To illustrate, we develop efficient algorithms for gate matrix layout. Although gate matrix has become an increasingly popular layout style for CMOS circuits, the combinatorial problem at the heart of this style is $\mathcal{NP}$-complete. Consequently, the research community has turned to efficient heuristic algorithms in an effort to find near-optimal layouts. We present a very different kind of approach. Focussing on three-track layouts as a starting point, our algorithms decide whether an input circuit has such a layout in linear time. They also find a three-track layout if any exist. Extensive experiments demonstrate the feasibility of our method. We discuss extensions of our approach to other problems of VLSI design.

---

* This research has been supported in part by the National Science Foundation under grant MIP–8919312 and by the Office of Naval Research under contract N00014–90–J–1855.

1. Introduction

VLSI layout abounds in $\mathcal{NP}$-Complete problems. Consequently, for practical expedience during layout optimization, attention is frequently restricted to fixed-parameter problem variants. For many of these variants, the existence of polynomial-time decision algorithms is known [FL1, R]. The general approach used is based on graph theory developed in [RS1, RS2]. Unfortunately, these algorithms are nonconstructive and hence impractical (see [FL3] for details). In this paper, we develop a new technique to obtain practical layout algorithms. We illustrate the effectiveness of our approach using gate matrix layout as our prototypical example.

Our paper is organized as follows. Section 2 gives an overview of our approach. In Section 3, we review the gate matrix layout problem. Section 4 presents our algorithm. Section 5 shows how our algorithm can be used to find a layout. Experimental results are presented in Section 6. In Section 7, we discuss a few conclusions and directions for future work.

2. Technical Background

It is known [RS2] that graphs are well-partially-ordered under the minor operation. It follows that many families of graphs (including those that we consider here), can be characterized by finite numbers of forbidden graphs, henceforth termed "obstructions." It is also known [RS1] that testing an arbitrary input graph for any fixed obstruction takes at most $O(n^3)$ time. Thus such families possess polynomial-time decision decision algorithms that work as follows: test the input for every obstruction.

Unfortunately, there can be no general scheme to enumerate all the obstructions to a given problem [FL3]. Moreover, even when all the obstructions to a problem are known, it is a daunting task to obtain efficient tests for each of them.

Thus we are motivated to search elsewhere for genuinely practical solutions. Our approach is to check for the exclusion of only a well-chosen subset of the complete obstruction set. With a careful selection of obstructions relevant to the problem at hand, we can often obtain surprisingly effective algorithms.

3. A Representative Problem

Gate matrix layout (GML) is a well-studied combinatorial problem that arises in several VLSI layout styles such as Gate Matrix and Weinberger Arrays. The input is a circuit consisting of m gates and n nets. The usual objective is to obtain a layout that minimizes the total area of silicon used. An instance of the gate matrix layout problem consists of a collection of nets (rows) $N = \{N_1, N_2, ..., N_n\}$ and their respective connections to a set of gates (columns) $G = \{G_1, G_2, ..., G_m\}$. We seek a permutation of the columns that minimizes the number of tracks required to lay out the circuit. More formally, we are given a $n \times m$ Boolean matrix M and a positive integer k, and are asked whether we can permute the columns of M so that, if in each row we change to $*$ every 0 lying between the row's leftmost and rightmost 1, then no column contains more than k 1s and $*$s.

Since GML is $\mathcal{NP}$-Complete [KF], most past research has centered on the development of fast heuristic algorithms [HPK, WHW etc.]. Most of these heuristics are simple greedy rules based on some form of the min-cut algorithm. Simulated annealing approaches have also been reported [IO]. All of these approaches are prone to producing markedly inferior layouts or to running unacceptably slowly. Alternately, an $O(m^2 2^m)$ time optimal dynamic programming algorithm is known [DKL].

In the fixed-parameter version of GML, denoted GML(k), the maximum number of tracks allowed for a layout is fixed at some constant k. The goal is to decide whether the input circuit has a layout using k or fewer tracks, and then to obtain a layout if possible. When $k = 1$, GML(k) is trivially solved. Using the algorithm presented in [BL], it is easy to obtain a two-track layout if possible, in $O(n^2)$ time. However, these algorithms do not extend to the cases when k is greater than two.

The smallest non-trivial case is when $k = 3$, and that is the case we address here. Although the consideration of three-track layouts may at first seem overly restrictive, it is known [LR, SS] that this class of graphs captures a large collection of circuits likely to be encountered in practice. (Note that [BK] also addresses the fixed-parameter problem and presents $O(n \log^2 n)$ algorithms. However, their algorithms remain impractical due to their immense constants of proportionality, which are exponential in k.)

4. A Fast Decision Algorithm

We have adopted the standard mapping of a circuit into a graph representing the interconnection relationships between the nets [DKL]. Each vertex in the graph corresponds to a net in the circuit. We test only for six out of the 110 obstructions to a three-track layout [KL]. The six obstructions are the graphs A, B, C, D, E and F, shown in Figure 1. These six graphs are the smallest obstructions to a three-track gate matrix layout.

Given the density of these six obstructions relative to the other 104 possibilities [KL], it is reasonable to expect that a circuit that cannot be laid out in three tracks will contain one of them. Our algorithm operates on this hypothesis.

4.1. Algorithm Description

Excluding K_4 implies that an input graph must be series-parallel. The well-known fact that every series-parallel graph has at least two vertices of

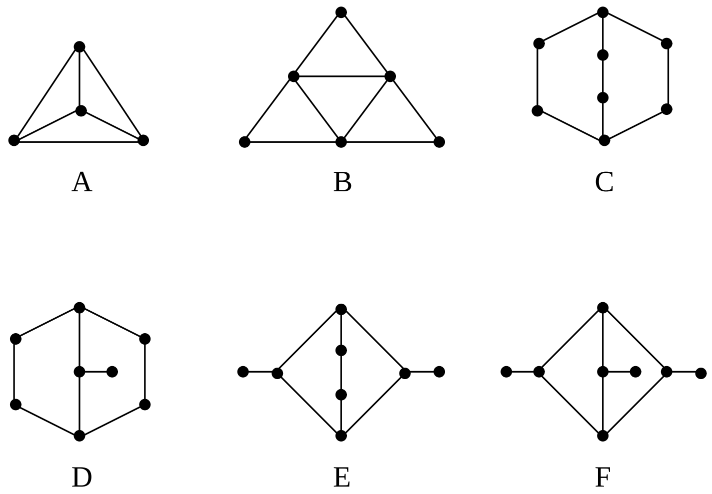

Figure 1. The six smallest obstructions to GML(3)

degree at most two is crucial to our method. Our algorithm proceeds in three steps, halting as soon as an obstruction is detected. In step 1, we look for graph A. In step 2, we look for graph B. In step 3, we look for graphs C, D, E and F. Each step involves finding a minimum degree vertex, processing it, and then eliminating or marking it. Algorithm TEST appears in pseudo-code form on the next page.

4.2 Proof of Correctness

Due to space limitations, we shall only sketch a proof of correctness of section 2 of algorithm TEST (where we look for graph B).

Lemma 1: *A biconnected graph H is a minor of a graph G if and only if H is a minor of a biconnected component of G.*

Proof Sketch: Use induction on the number of biconnected components of G. $\square$

Lemma 1 ensures that vertices of degree 1 can be deleted safely, since B is biconnected.

In the rest of this section, H is assumed to be a graph produced by

Algorithm TEST
Input : A simple graph G.
Output: YES, if G contains any of the graphs A, B, C, D, E and F.
NO otherwise.

```
begin procedure
0. Initialize.
      for each vertex a in G do
            unmark(a)
            if degree(a) ≤ 2 then set vertex-weight(a) = 0
            else set vertex-weight(a) = 1
      for each edge ab in G do set edge-weight(ab) = 1
1. Checking for graph A.
      repeat
            identify a minimum degree vertex v in G
            if degree(v) ≤ 1 then delete v
            else if degree(v) = 2 then
                  let w and x be the two neighbors of v
                  delete the edges vw and vx, and the vertex v
                  add the edge wx if it was not already present
      until G is empty or irreducible
      if G is nonempty then return YES
      else restore G and initialize it. Go to step 2
2. Checking for graph B.
      repeat
            while G contains a vertex v with degree ≤ 1 do delete v
            if G contains an unmarked vertex v with degree 2 then
                  mark v
                  let u and w be the two neighbors of v
                  if edge-weight(uv) = 3 and edge-weight(vw) = 3 then
                        go back to the start of the loop
                  if edge uw already exists then set edge-weight(uw) = 3
                  else
                        introduce edge uw
                        set edge-weight(uw) = max {edge-weight(uv), edge-weight(vw), 2}
                  delete the edges uv and vw, and the vertex v
      until G is empty or irreducible
      if G is nonempty then return YES
      else restore G and initialize it. Go to step 3
3. Checking for graphs C, D, E and F.
      repeat
            while G contains a vertex v with degree ≤ 1 do delete v
            if G contains an unmarked vertex v with degree 2 then
                  mark v
                  let u and w be the two neighbors of v
                  if edge-weight(uv) > 3 or edge-weight(vw) > 3 then
                        go back to the top of the loop
                  if edge uw already exists and
                     edge-weight(uv) + edge-weight(vw) vertex-weight(v) ≥ 3 then
                        if edge-weight(uw) > 3 then go back to the top of the loop
                        if edge-weight(uw) = 3 then set edge-weight(uw) = 4
                  if edge uw does not exist or edge-weight(uw) < 3 then
                        introduce the edge uw and set edge-weight(uw) =
                        min{edge-weight(uv) + edge-weight(vw) + vertex-weight(v), 3 }
                  delete the edges uv and vw, and the vertex v
      until G is empty or irreducible
      if G is nonempty then return YES else return NO
end procedure
```

applying a sequence of the operations in section 2 of TEST to a simple series-parallel graph G. We say H' is an intermediate graph in the transformation of G to H if G can be reduced to H' by applying a prefix of the sequence of operations needed to reduce G to H.

Lemma 2: *If there is an edge of weight 2 between a pair of vertices u and v in H, then there is a path of length two or more between u and v in G.*
Proof Sketch: Use the fact that an edge of weight 2 is introduced between u and v in H only if there is a third vertex w such that the edges uw and vw both exist in an intermediate graph H'. $\square$

Lemma 3: *If there is an edge of weight 3 between u and v in H, then u and v are end-points of some triangle in G.*
Proof Sketch: Use lemma 2 and the appropriate transformation in section 2 of TEST. $\square$

Lemma 4: *H is irreducible if and only if H has no vertices of degree one, and all edges incident on vertices of degree two in H have weight three.*
Proof Sketch: If H does not have these properties, then at least one more transformation is possible. $\square$

Theorem: *G contains graph B as a minor if and only if H is irreducible.*
Proof Sketch: Use lemmas 2, 3, 4 and induction on the number of vertices in H. $\square$

4.3 Time Complexity

The main operations in TEST are: searching for vertices of minimum degree, deleting vertices and edges, checking for the existence of specific edges, and adding edges. Using an uninitialized adjacency matrix representation for the graph [AHU], each of these operations can be performed in constant amortized time. Since a series-parallel graph with n vertices has a maximum of $2n - 3$ edges, each operation is used at most a linear number of times, guaranteeing that TEST runs in linear time.

5. A Fast Layout Algorithm

Now we describe how algorithm TEST can be used to find a layout. Again, the input circuit is transformed into a graph. We invoke TEST on the graph in order to decide whether a three-track layout is possible or not. If none of the six obstructions is found by TEST, then we proceed to find a layout using a technique known as "self-reduction" [FL2]. To accomplish this, we augment the graph in a systematic way. Each time we augment the graph, we invoke algorithm TEST in order to confirm that the augmentation is valid.

A valid augmentation is one that does not introduce any of the obstructions detected by algorithm TEST. If the augmentation is invalid, then we undo it and try a different one. We continue this process until no more valid augmentations are possible. At this point, the self-reduction is complete and the graph has the "consecutive ones property" [DKL]. It takes at most $O(n^2)$ calls to algorithm TEST to attain this property. It is easy to determine a layout from such a graph [BL].

Of course, the original graph may contain one of the 104 (sparser) obstructions that TEST ignores. More interestingly, it is possible that we might introduce an obstruction not detected by TEST. (In this case, our self-reduction would fail to yield a three-track layout when the input circuit actually has such a layout.) However, this happens extremely rarely, as our experiments reveal.

6. Experimental Results

Algorithm TEST and its self-reduction have been implemented. These methods are extremely fast in practice, indicating that the constants of proportionality are small. We first use TEST to decide whether three tracks are sufficient to layout a given circuit. If one of the six obstructions is found in the input graph, then we know that a three-track layout is not possible. On the other hand, if none of the six obstructions is found, then we assume that a layout is possible and self-reduce to find a layout.

Data sets consisting of a thousand graphs of a fixed size, each generated at random[1] with a specified edge probability were used as input to TEST. Results from our experiments are presented in the tables on the next page. The output of our decision algorithm and the layout algorithm are compared to the optimum solution found by the (extremely slow) algorithm of [DKL].

There are two reasons why our algorithm fails occasionally. The first reason is that the input graph contains an obstruction other than the six for which we test. In this case, the decision algorithm would say that a layout is possible, even though a layout does not exist. The second case is when we inadvertently introduce an extraneous obstruction during self-reduction. In this situation, we are unable to obtain a three-track layout for the input circuit even though such a layout is possible. From the tables, we infer that these failures occur only rarely. In fact, on the whole our methods construct a layout when any exist over 95% of the time!

---

[1]Of course, a randomly generated graph may not always represent a useful circuit. However, it is a valuable aid in our study.

Table 1: Experimental results with edge probability = 1/2

number of vertices	number of random graphs	number with a 3-track layout*	Results from Algorithm TEST: layout possible	layout found
8	1000	97	97	96
9	1000	12	12	12
10	1000	5	5	5
11	1000	1	1	1
12	1000	0	0	0

*determined by dynamic programming .

Table 2: Experimental results with edge probability = 1/3

number of vertices	number of random graphs	number with a 3-track layout	Results from Algorithm TEST: layout possible	layout found
8	1000	711	711	707
9	1000	454	455	444
10	1000	221	222	212
11	1000	94	95	85
12	1000	16	17	14
13	1000	6	6	6
14	1000	0	0	0

Table 3: Experimental results with edge probability = 1/4

number of vertices	number of random graphs	number with a 3-track layout	Results from Algorithm TEST: layout possible	layout found
8	1000	928	928	922
9	1000	789	790	773
10	1000	620	624	582
11	1000	449	453	405
12	1000	245	251	215
13	1000	121	125	104
14	1000	36	39	29
15	1000	14	15	11
16	1000	3	3	0
17	1000	0	0	0

7. Conclusions

Numerous problems in VLSI design are amenable to this approach. We have illustrated our method with but one example, gate matrix layout. For instance, linear time algorithms are possible for the minimum cut linear arrangement problem by extending and adding self-reduction to the immersion test of [BGLR].

We are currently generalizing our methods to more than three tracks. While three tracks are not sufficient to lay out arbitrarily large circuits, testing for three-track layouts alone is still a promising technique. It is often possible to layout functional cells in a few tracks in gate matrix style. One could also synthesize small to medium sized cells. Another possibility is to partition a large cell into smaller blocks, and layout each individual block separately [HW]. Some of these applications will be part of our future research.

There is clearly room for improvement in our approach. Better self-reduction algorithms and general tools to help develop fast tests for obstructions would be useful. We could improve the self-reduction scheme in at least two ways. First, we could try to avoid or at least minimize the likelihood of introducing new obstructions that are not detected by algorithm TEST. Secondly, we could minimize the number of calls to algorithm TEST. We are currently exploring some of these new avenues.

References

[AHU] A. V. Aho, J. E. Hopcroft and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[BGLR] H. D. Booth, R. Govindan, M. A. Langston and S. Ramachandramurthi, “Cutwidth Approximation in Linear Time,” *Proceedings, Great Lakes Symposium on VLSI* (1992), 70–73.

[BK] H. Bodlaender and T. Kloks, “Better Algorithms for the Pathwidth and Treewidth of Graphs,” Technical Report, University of Utrecht, The Netherlands (1990).

[BL] K. S. Booth and G. S. Leuker, “Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-tree Algorithms,” *J. of Computer and Systems Science* 13 (1976), 335–379.

[DKL] N. Deo, M. S. Krishnamoorthy and M. A. Langston, “Exact and Approximate Solutions for the Gate Matrix Layout Problem,” *IEEE Trans. on Computer-Aided Design* 6 (1987), 79–84.

[FL1] M. R. Fellows and M. A. Langston, “Nonconstructive Tools for Proving Polynomial-Time Decidability,” *J. of the ACM* 35 (1988), 727–739.

[FL2] ———, “Fast Search Algorithms for Layout Permutation Problems,” *Int'l J. Computer Aided VLSI Design* 3 (1991), 325–341.

[FL3] ———, “On Search, Decision and the Efficiency of Polynomial-Time Algorithms,” *Journal of Computer and System Sciences* (to appear). (A preliminary version of this paper can be found in Proceedings, ACM Symposium on Theory of Computing (1989), 501–512.)

[HPK] Y-S. Hong, K-H. Park, and M. Kim, “Heuristic Algorithms for Ordering the Columns in One-Dimensional Logic Arrays,” *IEEE Trans.*

on Computer-Aided Design 8 (1989), 547–562.

[HW] S. Huang and O. Wing, "Gate Matrix Partitioning," *IEEE Trans. on Computer-Aided Design* 8 (1989), 756–767.

[IO] M. J. Irwin and R. M. Owens, "A Comparison of Four Two-Dimensional Gate Matrix Tools," *Proceedings, Design Automation Conference* (1989), 698–701.

[KF] T. Kashiwabara and T. Fujisawa, "An NP-complete Problem on Interval Graphs," *Proceedings, International Symposium on Circuits and Systems* (1979), 82–83.

[KL] N. G. Kinnersley and M. A. Langston, "Obstruction Set Isolation for the Gate Matrix Layout Problem," University of Tennessee Computer Science Technical Report CS–91–126, 1991.

[LR] M. A. Langston and S. Ramachandramurthi, "Dense Layouts for Series-Parallel Circuits," *Proceedings, Great Lakes Symposium on VLSI* (1991), 14–17.

[R] B. A. Reed, "Finding Approximate Separators and Computing Tree Width Quickly," *Proceedings, ACM Symposium on Theory of Computing* (1992).

[RS1] N. Robertson and P. D. Seymour, "Graph Minors XIII. The Disjoint Paths Problem," *J. Comb. Th. Ser. B* (to appear).

[RS2] ———, "Graph Minors XVI. Wagner's Conjecture," *J. Comb. Th. Ser. B* (to appear).

[SS] C. C. Su and M. Sarrafzadeh, "Optimal Gate Matrix Layout of CMOS Functional Cells," *Integration, the VLSI Journal* 9 (1990), 3–23.

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