Cutwidth Approximation in Linear Time
(Extended Abstract)
Heather D. Booth, Rajeev Govindan, Michael A. Langston and Siddharthan Ramachandramurthi
Department of Computer Science
University of Tennessee
Knoxville, TN 37996

Abstract
Graph width metrics have been widely studied for their relevance to VLSI design. Examples include cutwidth, pathwidth, bandwidth and several others that arise in circuit layout. When the width is bounded, graphs that satisfy these metrics can often be recognized by finite lists of obstruction tests. One of the most foundational tests is to determine whether \(K_4\) is immersed in a graph. In this paper, we present for the first time a fast, practical algorithm to perform this test, and discuss its relevance to cutwidth and other metrics.

1 Introduction
Layout permutation problems abound in VLSI design theory. These can often be described in terms of graph width metrics. Familiar metrics include cutwidth, pathwidth, bandwidth and many others. Although deciding whether a graph satisfies one of these metrics is \(NP\)-complete in general, several of them can be decided in polynomial time as long as the relevant metric is bounded.

Cutwidth, for example, can in principle be decided in \(O(n^2)\) time for any fixed width using a finite but unknown list of immersion tests. Pathwidth, also amenable to this approach, can be decided analogously for any fixed width with a finite number of minor tests. Bandwidth, on the other hand, is not currently known to permit any such technique. We refer the reader to [4] for detailed information on this subject.

In this paper, we focus on cutwidth and the immersion order. The cutwidth of \(G = (V, E)\) is defined to be the maximum number of edges from \(E\) that must be cut between consecutive vertices in any linear layout of \(V\). A graph \(H\) is immersed in a graph \(G\) if a graph isomorphic to \(H\) can be obtained from \(G\) by a series of these two operations: taking a subgraph and lifting\(^1\) a pair of adjacent edges.

The aforementioned tests are for "obstruction" containment. An obstruction can be viewed in this setting as a forbidden graph. Thus, when one knows all obstructions to cutwidth \(k\), one knows a characterization for the family of graphs that have cutwidth \(k\) or less.

Unfortunately, very little is known about practical immersion tests or the nature of cutwidth obstructions. One problem is that multiple edges are important, making immersion tests much more "slippery" than, say, many well-known minor tests. As with other width metrics and other orders, however, complete graphs are obstructions. Testing for \(K_1\) and \(K_2\) are of course trivial. Testing for \(K_3\) is easy: \(K_3\) is immersed in any graph of order three or more unless the graph is a tree with no pair of multiple edges incident on a common vertex.

The first really difficult test, and the one we devise here, is that for \(K_4\). Ours is the first practical linear-time algorithm known for this task. Observe that \(K_4\) is an obstruction for cutwidth three, because any arrangement of its vertices on a line will require a cut of four edges. Moreover, those graphs with no immersed \(K_4\) are necessarily series-parallel\(^2\), a class of graphs frequently encountered in circuit layout. But \(K_4\) can be immersed in a series-parallel graph. As a simple example, consider the star graph with three rays, each ray with three edges:

In the next section we state relevant definitions and a few useful technical lemmas. In Section 3 we present our linear-time \(K_4\) test. In the final section we discuss its application in recognizing graphs of bounded cutwidth and other problems relevant to VLSI design.

2 Preliminaries

Let \(G = (V, E)\) denote a connected series-parallel multi-graph with \(n\) vertices and \(m\) edges. Without

\(^{1}\)A pair of edges \(uv\) and \(vw\), with \(u \neq v \neq w\), is lifted by deleting the edges \(uv\) and \(vw\) and adding the edge \(uw\).

\(^{2}\)A graph with no immersed \(K_4\) can contain no topological \(K_4\), which is one of several equivalent characterizations of series-parallel graphs.
A cut vertex (cut edge) of G is a vertex (edge) whose removal disconnects G. A pair of vertices is biconnected if they can be separated by no cut vertex. A biconnected component of G is the subgraph induced by a maximal subset of pairwise biconnected vertices. A pair of edges, neither of which is a cut edge, is a cut edge pair if the removal of both disconnects G. A pair of vertices is three-edge-connected if they can be separated by no cut edge or cut edge pair. A three-edge-connected component of G is a graph G' = (V', E') for which:

1) V' ⊆ V,
2) u, v ∈ V' ⇒ u and v are three-edge-connected in G,
3) u ∈ V' and v ∈ V - V' ⇒ u and v are not three-edge-connected in G, and
4) E' contains all edges induced by V' plus a set of virtual edges which join each pair of vertices in V' that appear in the same cut edge pair.

Several technical lemmas are required. We list three of them here. The first two permit us to consider only a graph's three-edge-connected components. The third lemma is well-known.

Lemma 1. K₄ is immersed in G if and only if K₄ is immersed in some three-edge-connected component of G.

Proof: The proof is by induction, and can be generalized to any three-edge-connected graph immersed in G. We omit the details. □

Lemma 2. Each three-edge-connected component of a series-parallel graph is series-parallel.

Proof: Omitted. □

Lemma 3. Any series-parallel graph contains at least two vertices with at most two neighbors.

3 The Algorithm

Our K₄ test, algorithm immerse, proceeds in two stages. We first invoke algorithm components, which breaks G into three-edge-connected components. Then we invoke algorithm test on each component until either a K₄ is encountered or all components have been eliminated (Lemma 1).

Algorithm components first finds the biconnected components of G using the method of [14]. The biconnected components are processed to eliminate all cut edge pairs. Each biconnected component of a series-parallel graph is a two-terminal series-parallel graph. These graphs can be defined recursively and have a simple structure [15]. We use properties of two-terminal series-parallel graphs to remove all cut edge pairs and add the appropriate virtual edges. Thus this algorithm exploits heavily the structure of series-parallel graphs, and hence is considerably simpler than a technique suggested [12] based on finding triconnected components with [6].

Pseudocode for components is lengthy and subtle. We omit it from this presentation.

Algorithm test is the heart of our method, and is described in the pidgin Algol that follows. Its input is series-parallel (Lemma 2). We proceed by examining vertices with at most two neighbors (Lemma 3). At each iteration, some such vertex is selected. The vertex is deleted (after deleting or lifting its incident edges) if we can determine that it is not contained in every copy of K₄ should K₄ be present. Otherwise, the vertex is marked. We assume all vertices are initially unmarked. As the algorithm progresses, a vertex may be marked then later unmarked again as its neighborhood changes.

**Theorem 1.** Algorithm immerse correctly decides whether K₄ is immersed in an input graph.

Proof: The analysis, especially for test, is based on a number of cases. We will not present it here. □

**Theorem 2.** Algorithm immerse runs in O(n) time.

Proof: Straightforward (recall that graphs of interest have a linear number of edges). □

Immersion containment may alternatively be viewed in the following manner. A graph H is immersed in a graph G if H can be injected into G so that (1) the images of the vertices of H are distinct vertices of G and (2) the images of the edges of H are pairwise edge disjoint paths in G connecting the appropriate image vertices. Such an image of H is termed its model with respect to G.

A natural extension to detecting an immersed graph is finding one of its models. Assuming K₄ is indeed immersed, algorithm test can easily be modified to return a model of K₄. The first step is to locate four image vertices. This is trivial and can be done in constant time. Six edge-disjoint paths are then located by lifting edges at vertices with only one or two neighbors and storing the sequence of lifts used. Details are forthcoming in the full version of this paper.
4 Discussion

The cutwidth metric has appeared in various layout settings (see, for example, [2, 3, 7, 8]). Integer weights are often placed on edges in these applications. Such weights can directly be modeled by multiple edges for the purpose of immersion testing. This is a significant feature of the immersion order; multiple edges can be ignored in the topological and minor orders. Deciding whether a graph has small cutwidth is a natural problem in the layout process. Not only are graphs that represent real circuits often series-parallel, but more generally they tend to be sparse (with at most a linear number of edges) and of bounded degree (due to limitations on porting or fan-in/fan-out). These are necessary, but not sufficient, conditions for bounded cutwidth.

A fast $K_4$ immersion test is a fundamental method. It can be employed to indicate whether a graph is likely to have cutwidth three (a $K_4$ minor test can be used analogously to predict pathwidth two [9]). Although the presence of an immersed $K_4$ guarantees that a graph cannot have cutwidth three, its absence merely approximates the cutwidth at three. Such an absence says nothing about how to find a layout of width three even if many exist. To solve this, our algorithm can be used in conjunction with previously-studied self-reduction techniques [1, 5] to try to find a layout in $O(n^2)$ time.

For completeness, we observe that there is a much less practical way to test for $K_4$ in linear time. It is possible in principle to use the method sketched in [16] to obtain a tree-decomposition of width two, and then to use the dynamic programming formulation of [13] for testing.
on the decomposition. This two step procedure runs in $O(n)$ time, although the constant of proportionality is extremely high. Cutwidth can even be decided directly, although the time complexity is exponential in the width [11].

In addition to cutwidth, other combinational problems relevant to circuit design can benefit from a fast $K_4$ test. For example, a variety of other load factor [4] problems can be decided by a finite list of immersion tests, including $K_4$. Another candidate, though one only indirectly approachable with this method, is graph bisection. Note that bounded cutwidth is a sufficient, but not necessary, condition for bounded graph bisection. For problems such as these, there is interest in devising fast tests for other important graphs [10]. It will be interesting to see how quickly this general approach makes the transition from theory to practice.

References


