1. Introduction

The problem of laying out VLSI circuits has led to many interesting graph theoretic problems. A circuit is abstracted into a graph whose nodes represent the processing elements and whose edges represent their interconnections. The design methodology in the VLSI domain determines in each case the rules of the game in the graph layout problem. Several graph problems thus arising from VLSI (as well as other areas) ask for the placement of the nodes of the graph on a line to minimize some cost function. In this paper we discuss two such problems that involve linear layouts of graphs.

Some approaches to VLSI design simplify the layout task by placing the circuit elements in rows or on a line [F, FK, W]. Figure 1 shows such a linear layout of a circuit; the boxes stand for the circuit elements. The elements are placed along a horizontal line, and their interconnecting wires run along parallel tracks. The cutwidth problem is the problem of finding an ordering of the elements which minimizes the number of tracks, and thus also the area needed for the wiring. In Section 2 we review some of the algorithmic results on the cutwidth problem.

A recent approach to the design of fault-tolerant VLSI processor arrays reduces also the problem to one dimension (see [R, CLR1, CLR2] and Rosenberg's paper in this workshop). The processors are arranged (physically or logically) on a line, and are tested to determine which ones are good and which are faulty. The good processors are interconnected via "bundles" of wires running parallel to the line. Each bundle functions like a stack: Scan the line from left to right, and suppose that a good processor $u$ wants to connect to some processor $v$ to its right. At $u$, the connection $(u,v)$ is pushed into one of the stack-bundles; that is, $(u,v)$ occupies the bottom wire of the bundle, while the other connections that are currently in this bundle are shifted up one place. At $v$, the connection $(u,v)$ is popped from the stack. The problem is to realize a desired interconnection pattern using the minimum number of stacks. This can be stated as the problem of embedding a graph in a book with the minimum number of pages. The nodes of the graph are embedded on the spine of the book, and its edges are drawn on the pages without any intersections.

In Section 3 we discuss the book embedding problem, and sketch an algorithm which embeds planar graphs in four pages. In Section 4 we look at the interaction between cutwidth and number of pages.
2. Cutwidth

We are given a graph $G = (N,E)$ with $n$ nodes. A (linear) layout $L$ of $G$ is an ordering of $N$; i.e., a one-to-one function from $N$ to $\{1, \ldots, n\}$. The cut of $L$ at a point $p$ of the real line is the number of edges $(u,v)$ of $G$ with $L(u) < p < L(v)$; i.e., the number of edges that connect nodes (strictly) left of $p$ to nodes (strictly) right of $p$. The cutwidth of $L$, denoted $\gamma(L)$, is the maximum cut of $L$ over all real points. The cutwidth $\gamma(G)$ of a graph $G$ is the minimum cutwidth of a layout for $G$. In Figure 2 we show the linear layout of the graph corresponding to the circuit of Fig. 1; the graph is the complete binary tree of height 2 with root $a$. This layout has cutwidth 2, which is the minimum possible for this particular graph. The cutwidth of a graph is equal to the minimum number of tracks needed to route the interconnections of the corresponding circuit.

![Figure 2](image)

The cutwidth problem is NP-complete for general graphs [Ga], even when the maximum degree is 3 [MPS]. For any fixed $k$ it is possible to determine in time $O(n^{k+1})$ if the cutwidth of a graph is at most $k$ [MS]. In the case of trees Chung, Makedon, Sudborough and Turner developed first an algorithm which runs in polynomial time if the maximum degree $d$ is fixed [CMST]. An algorithm which solves the cutwidth problem for arbitrary trees in $O(n \log n)$ time was presented in [Y1]. The reason that trees are easier, is that they have a nice recursive structure so that a dynamic programming algorithm can often be used. The tree is rooted at some node, and optimal solutions for the subtrees are computed in a bottom-up manner. The problem then reduces to a one level problem. Suppose that we are at a node $v$ and we have already computed optimal solutions for the subtrees rooted at the children of $v$; we must combine these solutions to produce an optimal solution for the subtree rooted at $v$.

The trouble with the cutwidth problem is that the "obvious" method does not work. It is tempting to place the nodes of the different subtrees consecutively and in disjoint intervals. Thus, in the case of a complete binary tree, we would first place the nodes of the left subtree, then the root and then the nodes of the right subtree (ordered recursively in an optimal way). This layout corresponds to an inorder traversal of the complete binary tree and has cutwidth equal to its height. However, there is another layout with cutwidth about half of that (\[ \lceil h/2 \rceil + 1, \text{where } h \text{ is the height } \]) [L]. The optimal strategy works at even heights $2k$ like the obvious method: left subtree - root - right subtree. But at odd heights $2k+1$ it works as Figure 2 suggests, where $d, e, f, g$ stand for complete binary trees of height $2k-1$; that is, left subtree - left subtree of right son - root - right son - right subtree of right son.

It is this possible interlacing of the root and the various subtrees that causes the difficulties. The algorithm in [Y1] still uses dynamic programming. However, in order to optimize the cutwidth of the whole tree, it has to find layouts for the subtrees that are optimal not only with respect to cutwidth, but with respect to many more parameters - a linear number of them in the worst case. Given the difficulty of the cutwidth problem for trees, it does not seem that one can expect polynomial time solutions for significantly larger classes of graphs. It may be the case that the algorithm could be extended to outerplanar graphs (with quite a bit of effort, if at all possible), but probably not much beyond that.

Problem. Can one find good approximations to the cutwidth of planar graphs (at least of fixed degree $d$)?

Approximating the cutwidth is related to the problem of finding good approximations to the bisection width or to separation of graphs into roughly equal parts. We say that a class $C$ of graphs has $f(n)$ node (edge) separators, if for every graph $G$ in $C$ with $n$ nodes there is a set of $f(n)$ nodes (respectively, edges) whose deletion breaks the graph into two roughly equal parts (say, each part has no more than $2/3$ of the nodes), and the parts themselves are in $C$. Trees have 1 node separators, outerplanar graphs have 2 node, and planar graphs have $O(\sqrt{n})$ node separators. Furthermore, such separators can be found efficiently. Thus, trees and outerplanar graphs with fixed degree