Discrete Applied Mathematics 23 (1989) 243–265
North-Holland



# ON MINIMIZING WIDTH IN LINEAR LAYOUTS*

Fillia MAKEDON** and Ivan Hal SUDBOROUGH**
*Computer Science Program, University of Texas at Dallas, Richardson, TX 75083-0688, USA*

Received 22 December 1983
Revised 30 August 1988

Let $G$ be a finite undirected graph and let $cw(G)$, $s(G)$ and $b(G)$ denote the cutwidth, search number, and bandwidth of $G$, respectively. Characterizations of graphs with cutwidth 2 and cutwidth 3 are described. Cutwidth is related to search number and bandwidth by the inequalities: $s(G) \leq cw(G) \leq \lfloor \frac{1}{2}\deg(G) \rfloor \cdot (s(G)-1)+1$ and $cw(G) \leq \lfloor \frac{1}{2}\deg(G) \rfloor \cdot b(G)+1$, where $\deg(G)$ denotes the maximum degree of any vertex in the graph $G$. It follows as an interesting corollary that search number and cutwidth are identical for graphs with maximum vertex degree three. It is also shown that these inequalities are almost the best possible. Finally, a dynamic programming algorithm is described that determines, for each fixed $k \geq 2$, whether $cw(G) \leq k$ or not in $O(n^{k-1})$ steps. Thus, it follows that graphs with cutwidth 2 can be recognized in linear time.

## 1. Introduction

In 1967 Arnold Weinberger [18] described a method for large scale integration of MOS complex logic circuits which is often used for the purposes of automation. The circuit board or chip is produced with a single type of gate, e.g. NOR or NAND gates; all gates are placed in horizontal rows with uniformly spaced intervals between successive gates. The chip is "personalized" to represent a particular circuit by the way in which the gates are interconnected. This approach using Weinberger arrays, also called "gate matrices" and "uncommitted logic arrays", has been the subject of several papers in the literature [1, 3, 9, 12, 15, 17, 20]. An objective of this approach is to find an optimal arrangement of the circuit elements so that the number of horizontal tracks needed for their interconnection is minimized.

An equivalent problem has been studied in graph theory. Let $G=(V,E)$ be a finite undirected graph. A *linear layout of* $G$ is a one-to-one function $L$ mapping the vertices of $G$ to integers. That is, $L$ is simply a numbering of the vertices. The *cutwidth of* $G$ *under the linear layout* $L$, denoted by $cw(G,L)$, is the maximum over all integers $i$, of the number of edges that connect vertices assigned to integers smaller than $i$ with vertices assigned to integers at least as large as $i$. That is, $cw(G,L)$ is the

* A preliminary version of these results appear in: Proceedings 10th International Conference on Automata, Languages and Programming (ICALP), Barcelona, Spain (1983).

** Supported in part by NSF grant MCS 81-09280. A portion of this work was done while the authors were visiting the National Technical University of Athens, Greece.

maximum number of edges connecting vertices on opposite sides of any of the "gaps" between successive vertices in the linear layout $L$. The *cutwidth of* $G$, denoted by $cw(G)$, is $\min\{cw(G,L) \mid L \text{ is a linear layout of } G\}$. The problem in graph theory, called the min cut linear arrangement problem, is the following:

**Min cut linear arrangement (min cut) problem.**
*Input.* A finite undirected graph $G=(V,E)$ and a positive integer $k$,
*Question.* Is $cw(G) \leq k$?

An example is shown in Fig. 1. That the min cut problem is equivalent to the problem of minimizing the number of "tracks" or "channels" in a horizontal circuit layout is shown in [5].

The min cut problem is known to be NP-complete [5, 16]; in fact, it is known to remain NP-complete for graphs with maximum vertex degree 3 [10]. Polynomial time algorithms are known for the min cut problem on trees. First, an $O(n \log n)$ algorithm was described by Lengauer [8] which produced a layout $L$ for any tree $T$ such that $cw(T,L) \leq 2 \cdot cw(T)$. Secondly, for each $d \geq 3$, an $O(n \log^{d-2} n)$ algorithm was described [2] which solved the min cut problem for the class of trees with maximum vertex degree $d$. Most recently, an $O(n \log n)$ algorithm has been announced by Yannakakis [19] to solve the min cut problem for arbitrary trees.

In [2] it was shown that the cutwidth of a tree with maximum vertex degree 3 is identical to its search number. The search number of a graph $G$, denoted by $s(G)$, has been studied in [11, 14]. In Section 3 of this paper we extend this relationship to show that search number and cutwidth are, in fact, identical for all graphs with maximum vertex degree 3. Furthermore, we show that, for any graph $G$,

$$s(G) \leq cw(G) \leq \lfloor \tfrac{1}{2}\deg(G) \rfloor \cdot (s(G)-1)+1,$$

where $\deg(G)$ denotes the maximum degree of any vertex in $G$.

In Section 2 we give characterizations of graphs having cutwidth 2 and cutwidth 3. These characterizations strongly suggest linear time algorithms for determining if a graph $G$ has cutwidth 2 or cutwidth 3. We do not, however, explicitly give such algorithms. Previous results in the literature give characterizations of graphs with search number 2 and 3 [11] and a linear time algorithm for recognizing graphs with bandwidth 2 [4].

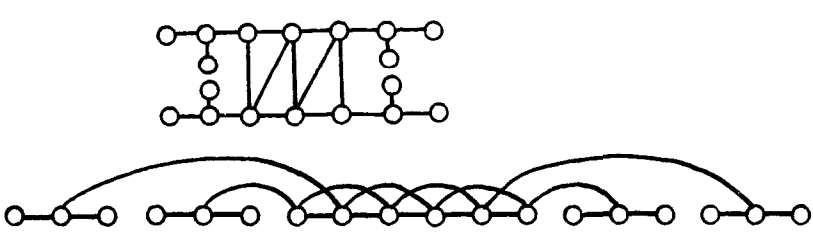

Fig. 1. A graph with cutwidth three (top) and a cutwidth three layout of this graph (bottom).

In Section 4 we give explicitly a dynamic programming algorithm which determines, for each fixed $k \geq 2$, whether $cw(G) \leq k$ or not in at most $O(n^{k-1})$ steps. This improves a previous dynamic programming algorithm [6] which required $O(n^k)$ steps. In particular, the dynamic programming algorithm given in Section 4, explicitly shows that graphs with cutwidth 2 can be recognized in linear time.

We shall assume throughout that the graphs considered are connected. It is easily seen that an arbitrary graph $G$ has cutwidth $k$ if and only if each of its connected components has cutwidth $k$. Therefore, our results, while at first seemingly limited to the connected components of a graph, in fact give the corresponding result for the entire graph.

## 2. Characterizing graphs with cutwidth 2 and 3

We describe characterizations of graphs with cutwidth 2 and cutwidth 3. Let us first consider various operations on a graph and their effect on cutwidth. The first operation is called "node splitting". That is, let $x$ be a vertex in a graph $G$ that is incident to a set of edges $E(x)$. Consider the partition of the set $E(x)$ into any two sets $E_{x1}$ and $E_{x2}$. Construct the graph $G'$, which has the same set of vertices as $G$, except that vertex $x$ has been replaced by two distinct vertices $x_1$ and $x_2$, and which has the same edges as $G$, except that the edges in $E_{x1}$ are made incident to $x_1$ and the edges in $E_{x2}$ are made incident to $x_2$. An example of the operation of node splitting is shown in Fig. 2.

It is straightforward to show that, if $G'$ is obtained from $G$ by node splitting, then $cw(G') \leq cw(G)$. We note also that the inverse operation of "coalescing" vertices cannot decrease the cutwidth, provided that it is understood that, when $x$ and $y$ are coalesced and they are joined by an edge in the original graph, the new graph has a self loop on the coalesced vertex. (The cutwidth of a graph with a self loop on a vertex $x$ is defined to be the same cutwidth as the graph in which a single degree 2 vertex is inserted into the self loop.)

Next consider the operation of "edge subdivision". That is, let $e$ be an edge in a graph $G$ connecting vertices $x$ and $y$. Construct the new graph $G'$ from $G$ by adding a new vertex $z$, and replacing the edge $e$ with two edges, one connecting $x$ and $z$ and one connecting $z$ and $y$. It is straightforward to see that $G'$ has the same cutwidth as $G$.

Similarly, the operation of "reduction", which is the deletion of degree 2 vertices

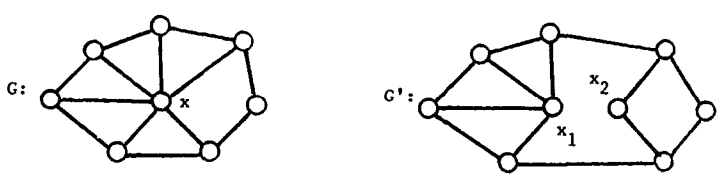


Fig. 2. $G'$ is obtained from $G$ by node splitting.

from an edge, does not change the cutwidth of a graph. That is, if $G$ has a degree 2 vertex $x$ adjacent to vertices $y$ and $z$, then $G'$, which is obtained by deleting $x$ and its incident edges and joining $y$ and $z$ together by a new edge, is obtained by a single reduction operation. The *reduction of* $G$ is the graph obtained from $G$ by applying all possible reduction operations.

Finally, the operation of deleting an edge can, clearly, not increase the cutwidth of a graph. So, if $G'$ is obtained from $G$ by the deletion of some edge, then $cw(G') \leq cw(G)$.

**Theorem 2.1.** *Let $G=(V,E)$ be a finite undirected graph. The following statements are equivalent:*

(1) *$G$ has cutwidth at most 2.*

(2) *$G$ does not contain a homeomorphic image of any of the graphs shown in Fig. 3 or of any of the graphs that can be obtained from any of these graphs by coalescing vertices.*

(3) *The reduced graph $G'$ of $G$ has at most vertex degree 4 and consists of a sequence of vertices $a_1, a_2, \ldots, a_r$ (for some $r \geq 1$) such that, for all $i$ $(1 \leq i < r)$, $a_i$ is joined to $a_{i+1}$ by either one or two edges, together with the following attachments:* (a) *single edges and* (b) *a self loop added to one of the vertices $a_1$, $a_2$, $a_{r-1}$, or $a_r$, provided that, if a self loop is attached to $a_2$ ($a_{r-1}$), then $a_1$ ($a_r$, respectively) has degree one.*

**Proof.** (1) → (2) It is straightforward to verify that all of the graphs shown in Fig. 3 have cutwidth larger than 2. Since coalescing vertices can not decrease cutwidth, graphs formed from these graphs by coalescing vertices can not have cutwidth 2. Also, no graph that is a homeomorphic image of these graphs can have cutwidth 2. Since the cutwidth of a graph is at least as large as the cutwidth of any of its subgraphs, if $G$ has a subgraph that is a homeomorphic image of any of these graphs, then $G$ cannot have cutwidth 2.

(2) → (3) We assume that $G$ does not contain a subgraph that is a homeomorphic image of any of the five subgraphs shown in Fig. 3 or of any graph formed from these graphs by coalescing vertices. Let $G'$ be the reduction of $G$, so that $G'$ has no degree 2 vertices. We will show that $G'$ consists of a chain of vertices $a_1, a_2, \ldots, a_r$ (some $r \geq 1$), with $a_i$ attached to $a_{i+1}$ by one or two edges, for all $i$ $(1 \leq i < r)$,

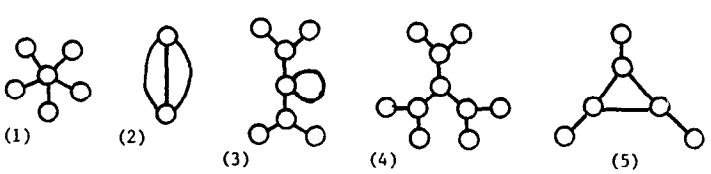


Fig. 3. Graphs that are forbidden components of any cutwidth 2 graph.

together with the type of attachments indicated. We note that such a chain must contain all vertices in $G'$ with degree greater than 2.

Suppose that there were no chain that contained all vertices in $G'$ with degree at least 3. Then, there must be a vertex $x$ and three vertex disjoint paths to vertices, say $a$, $b$, and $c$, with degree at least three. We observe that no cycle can contain two of the vertices $a$, $b$, $c$ and also the vertex $x$, since then the graph $G'$ would contain a homeomorphic image of the fifth graph in Fig. 3 or one obtained from it by coalescing vertices. But, if there is no cycle and there is a vertex $x$ with three disjoint paths to vertices with degree at least three, then $G'$ would contain a subgraph that is a homeomorphic image of the fourth graph in Fig. 3. So, in fact, there can be no such vertex $x$, and it must be possible to construct a chain $C$ containing all vertices with degree greater than two. Furthermore, since $G'$ does not contain any degree 2 vertices, all vertices not part of the chain must have degree one. It follows that $G'$ must be the chain $C$ together with single edges or self loops added to the vertices of this chain.

Suppose that a self loop were attached to a vertex $x$ in this chain $C$ such that $x$ had two neighbors with degree 3. Then, $G'$ would contain a subgraph that is a homeomorphic image of the third graph in Fig. 3 or a graph formed from it by coalescing vertices. So, in fact, the only self loops must be attached to vertices at the ends of the chain as specified. Clearly the successive vertices of the chain $C$ cannot be connected by more than two edges, for then the second graph of Fig. 3 would be a subgraph of $G'$. Finally, no vertex can have degree greater than four, for then the first graph of Fig. 3 would be a subgraph of $G'$.

(3) $\rightarrow$ (1) Let $G$ be a graph whose reduction $G'$ satisfies the conditions indicated in the third statement. To show that $G$ has cutwidth 2 it is sufficient to show that $G'$ has cutwidth 2. This is straightforward. One creates a layout in which the chain vertices $a_1, a_2, \ldots, a_r$ (for some $r \geq 1$) are laid out in the order they appear in the sequence and then "folds in" the edges and self loops attached to the chain vertices in a straightforward manner. □

Let $G$ be a reduced graph with cutwidth 2. By Theorem 2.1 there is a chain of vertices $a_1, a_2, \ldots, a_r$ (some $r \geq 1$) such that, for all $i$ $(1 \leq i < r)$ $a_i$ is connected to $a_{i+1}$ by one or two edges such that $G$ consists of the chain $C$ together with single edges or self loops added to the vertices in the stated manner. An "endpoint" of $G$ is a vertex such that, for some chain $C$, as above, it is (1) either the first or last vertex in the sequence, when there is no self loop at that end of the sequence, or (2) a new degree 2 vertex inserted into a self loop of $G$. That is, an endpoint of $G$ is a vertex that can be the first or the last vertex of $G$ in a cutwidth 2 layout.

**Definition 2.2.** A graph $G$ is *outerplanar* if it has a planar embedding in which a single face includes all of its nodes. The edges of that face are called *sides* and the remaining edges are called *chords*. A graph $G$ is *biconnected* if the removal of any single vertex from $G$ (and all of its incident edges) leaves a connected graph. A bi-

connected outerplanar graph $G$ satisfies the *collinear chord* property if it is possible to add some chords to $G$ (possibly zero) forming an outerplanar graph $G'$ in which all the chords form a simple chain.

In other words, a biconnected outerplanar graph $G$ satisfies the collinear chord property if in an outerplanar representation of $G$ one can draw a line, never entering the external region or picking up the pencil, which passes through all of the chords of $G$ such that one part of the line never crosses or touches another part of the line. Examples of a biconnected outerplanar graph with the collinear chord property and a biconnected outerplanar graph that does not satisfy the collinear chord property are given in Fig. 4.

**Lemma 2.3.** *Let $G$ be a finite biconnected graph. The following statements are equivalent*:

(1) $cw(G) \leq 3$.
(2) *The reduction of $G$ is outerplanar and satisfies the collinear chord property.*

**Proof.** (1) → (2) Let $G$ have cutwidth 3. Let $G'$ be the reduction of $G$. Consider a layout $L$ of $G'$ such that $cw(G', L) \leq 3$. Let $x$ and $y$ be the first and last vertices of $G'$ under the layout $L$, respectively. Since $G'$ is biconnected there must be two vertex disjoint paths, say $P_1$ and $P_2$, connecting $x$ and $y$. The vertices in these two paths must be positioned in such a way by the layout $L$ such that if $u$ is closer than $v$ to the vertex $x$ by the path $P_i$ $(i = 1, 2)$, then $u$ is before $v$ in the layout. If this were not true, then $L$ could not be a cutwidth 3 layout of $G'$. Furthermore, in every cut of $G'$ between $x$ and $y$, there are at least two edges, one contributed by the path $P_1$ and one by the path $P_2$. So, if we delete the edges in these two paths from $G'$, the result must be a cutwidth one graph. (It need not be connected, however.) First, observe that every vertex in $G'$ lies on one of the two paths $P_1$ or $P_2$. For suppose a vertex $x$ did not lie on either path. Such a vertex must have degree one, since there are no vertices of degree two in a reduced graph and if the degree was three or more, then the cut adjacent to this vertex would be at least of size four. However, no vertex in a biconnected graph with at least three vertices can have degree one. So, it follows that every vertex in $G'$ lies on one of the two paths: $P_1$ or $P_2$.

Next, we observe that when the edges from the two paths $P_1$ and $P_2$ are deleted from $G'$ the remaining edges connect vertices which are adjacent in the layout $L$.


Fig. 4. A biconnected outerplanar graph (a) satisfying the collinear chord property and (b) not satisfying the collinear chord property.

For otherwise, if a remaining edge connected vertices $u$ and $v$ and passed over a vertex $w$, then the vertex $w$ must have had degree 2 in the graph $G'$ and this contradicts the fact that $G'$ is reduced.

Now consider the following outerplanar representation of $G'$. The vertices of $G'$ are laid out along a line in the order specified by the layout $L$. The edges in the path $P_1$ are drawn above the line, the edges in the path $P_2$ are drawn below the line, and all of the remaining edges are positioned on the line containing the vertices. This is, in fact, an outerplanar layout, since all of the vertices of $G'$ are on the external region and none of the edges intersect except at the ends. The edges in the line containing the vertices are the chords. Furthermore, it is evident that $G'$ satisfies the collinear chord property, since one can draw a line starting from the leftmost vertex in the layout and proceeding through the line of vertices to the rightmost vertex. This line must, as we have seen, pass through all of the chords. So, if $G$ has cutwidth 3, then the reduced graph $G'$ must be outerplanar and satisfy the collinear chord property.

(2)→(1) For the other direction, let $G$ be a biconnected graph whose reduction $G'$ has an outerplanar representation and satisfies the collinear chord property. We need to show that $G'$ has cutwidth 3, since $G$ and its reduction $G'$ have the same cutwidth. As $G'$ satisfies the collinear chord property there is a chain $a_1, a_2, \ldots, a_p$ (some $p \geq 1$) such that, for every chord $c$ in $G'$, there is an $i$ $(1 \leq i < p)$ such that $\{a_i, a_{i+1}\} = c$. This chain must, in fact, include every vertex in $G'$, since every vertex in $G'$ is incident to a chord, i.e. it must have degree $\geq 3$. Furthermore, no vertex is listed twice in this chain by the collinear chord property. So, define the layout $L$ that maps the vertex $a_i$ to the integer $i$, for all $i$ $(1 \leq i \leq p)$. $G'$ must have cutwidth at most three under the layout $L$, since the sides of $G'$ (from its outerplanar representation) add only two additional edges to every cut across the layout $L$. □

Let $G$ be an arbitrary graph. A biconnected component of $G$ is "simple" if it consists of a single edge; otherwise, the biconnected component is said to be "nonsimple". An arbitrary vertex of $G$ is "simple" if it does not belong to any nonsimple biconnected component; it is "semisimple" if it belongs to a simple biconnected component and to a nonsimple component.

**Theorem 2.4.** *Let $G$ be a finite undirected graph. $G$ has cutwidth at most three if and only if $G$ can be obtained by the operations of node splitting, edge subdivision, reduction, and edge deletion from a chain $C = C_1, \ldots, C_m$ $(m \geq 1)$ of biconnected graphs, together with cutwidth two graphs attached to some of its vertices, such that the following properties are satisfied:*

(1) *For all $i$ $(1 \leq i < m)$, $C_i$ and $C_{i+1}$ share an articulation point $a_i$.*

(2) *For all $i$ $(1 \leq i \leq m)$, $C_i$ satisfies the conditions of Lemma 2.3; in fact, all of the chords in $C_i$ are part of a line that can be drawn from $a_{i-1}$ to $a_i$ ($a_0$ and $a_m$ are vertices that are chosen in $C_1$ and $C_m$, respectively).*

(3) *For all $i$ $(1 \leq i < m)$, if $a_i$ is simple, then a single cutwidth 2 graph can be at-*

*tached to $a_i$ and, if $a_i$ is semisimple, then a single cutwidth 2 graph can be attached by an endpoint to $a_i$.*

(4) *No attachments of cutwidth 2 graphs can be made to C except those allowed in* (3).

**Proof.** First we show that, if $G$ can be obtained from such a chain and attachments by the stated operations, then $G$ has cutwidth at most 3. In fact, since the operations do not increase the cutwidth of a graph, it is sufficient to show that such a chain with the indicated attachments always has cutwidth at most three. This is relatively straightforward. That is, one first lays out the chain of biconnected graphs. It is clear, since each biconnected graph satisfies the collinear chord property of Lemma 2.3 and the chain of chords can be drawn from one articulation point to the next, that this chain of biconnected graphs can be laid out with cutwidth 3. Thus, it is sufficient to show that all the indicated attachments can be "folded in" without making the cutwidth greater than three. If $a_i$ is a simple articulation point, i.e. it is a vertex shared by two single edges in the chain $C$, then clearly the indicated cutwidth 2 attachment can be laid out with just these two edges passing over and the resulting cuts between successive vertices have 3 edges. Similarly, if $a_i$ is a semisimple articulation point, i.e. it is a vertex shared by a simple edge and a nonsimple biconnected component, then a cutwidth 2 graph attached by an endpoint to $a_i$ can be laid out under the simple edge with the cutwidth total at most three. That is, since the cutwidth 2 graph is attached by its endpoint, it can be laid out with cutwidth 2 and its attached vertex at one end of its layout. So, it follows that any chain of biconnected graphs with the indicated attachments has a layout with cutwidth at most 3.

Conversely, let $G$ be a graph with cutwidth at most 3. Let $L$ be a layout of $G$ such that $cw(G, L) \leq 3$. Let $A$ and $B$ be the vertices that are first and last, respectively, under the layout $L$. There must be a path connecting $A$ to $B$. Let $P$ be such a path. Let $C_1, C_2, \ldots, C_m$ be the biconnected components of $G$ that share at least two vertices with path $P$ (for some $m \geq 1$) and let $a_i$, for all $i$ $(1 \leq i < m)$, be the articulation point that is part of the components $C_i$ and $C_{i+1}$.

If all of the edges in the components $C_1, C_2, \ldots, C_m$ are deleted, the remaining graph must have cutwidth at most two, since there is at least one edge from the path $P$ in each of the cuts in the linear layout $L$. So, all attachments to this chain of components must have cutwidth at most two.

Consider now the attachments one of the nonsimple biconnected components in this chain may have. Let $C_i$ be any such nonsimple component. $C_i$ can have several graphs with cutwidth one attached to it; however, at most two graphs with cutwidth two can be attached. That is, from the leftmost vertex of $C_i$ to the rightmost vertex of $C_i$ (in the layout $L$) there are at least two edges that are part of $C_i$ in each successive cut. Thus, at most two cutwidth two attachments are possible: one could be laid out before the leftmost vertex of $C_i$ and one laid out after the rightmost vertex of $C_i$. We want now to show that these cutwidth two attachments to $C_i$ can be ob-

tained by node splitting and reduction operations from cutwidth 2 graphs attached by their endpoints to either $a_{i-1}$ or $a_i$.

Let $a_i$ be one of the nonsimple articulation points. There can be at most one edge passing over $a_i$. (An edge "passes over" a vertex $x$ in a linear layout if it connects a vertex placed to the right of $x$ with a vertex placed to the left of $x$.) If there were two edges passing over $a_i$ then $G$ would have cutwidth at least 4, since $a_i$ has degree at least 3. For all $i$ $(1 \leq i < m)$, if there is an edge passing over $a_i$ in the layout $L$, then subdivide this edge by inserting a new degree 2 vertex and then coalescing this new degree 2 vertex with $a_i$. The resulting graph still has cutwidth at most 3. One can, of course, obtain the original graph back again by the inverse operations: node splitting and reduction.

By doing this to each nonsimple articulation point $a_i$ we create a graph $G'$ with cutwidth at most 3 such that all cutwidth 2 attachments are attached to either simple articulation points or semisimple articulation points. We note that if a cutwidth 2 graph is attached to a vertex in a nonsimple component $C_i$ through an edge $e$ and the component $C_{i-1}$ is a nonsimple component with no chords, then it may be possible to lay out the attachment to the left of both $C_i$ and $C_{i-1}$. If this is the case in the layout $L$, then the edge $e$ will be subdivided with at least two degree 2 vertices, one coalesced with $a_i$, and one coalesced with $a_{i-1}$. See Fig. 5 for an example.

A cutwidth 2 attachment in $G'$, which was formerly a cutwidth 2 attachment to a nonsimple component of $G$, must be attached by an endpoint to a semisimple articulation point. That is, it must be laid out by the layout $L$ either entirely to the left or entirely to the right of every nonsimple component. (Only one edge passes over any articulation point in $G$ and it has been subdivided and coalesced with the articulation point in $G'$. Thus, no edges pass over these nonsimple articulation points in $G'$. Clearly, no cutwidth 2 graph can be laid out in positions between the leftmost and rightmost vertices of a nonsimple biconnected component and still have cutwidth 3.)

All of the cutwidth one attachments to the nonsimple components of $G'$ and all of the chords of these components must satisfy a collinear property. That is, for all $i$ $(1 \leq i < m)$, if $C_i$ is a nonsimple component in $G'$, then from the leftmost vertex

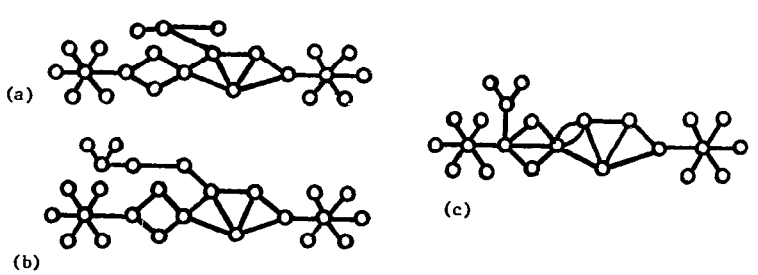


Fig. 5. (a) A graph showing a nonsimple component and cutwidth 2 attachment; (b) subdividing the edge of this attachment; and (c) coalescing the added degree 2 vertices with the nonsimple articulation points.

of $C_i$ to the rightmost vertex of $C_i$ in the layout there are at least two edges from $C_i$. Following the same argument used in Lemma 2.3 it can be seen that all the chords of this component and all the degree one attachments must satisfy a collinearity property. So, one may coalesce vertices and perform reductions and arrive at a component that is outerplanar and satisfies the collinear chord property. By the inverse of these operations, namely node splitting and edge subdivision, one may reconstruct the original component $C_i$.

Thus, we have shown that, for any graph $G$ with cutwidth 3, there is a graph $G'$, consisting of a chain of biconnected graphs and attachments with cutwidth at most 2, satisfying all of the properties indicated, such that $G$ can be obtained from $G'$ by node splitting, edge subdivision, reduction and edge deletion. □

Some graphs with cutwidth 3 are shown in Fig. 6; some with cutwidth greater than 3 are shown in Fig. 7.

## 3. Relating cutwidth to search number

The search number of a graph has been defined and the complexity of computing it has been discussed in [2, 11, 13, 14]. The search number of a graph $G$, denoted by $s(G)$, is the minimum number of searchers needed to guarantee catching a fugitive who is lurking about on the edges of $G$. The fugitive is assumed to have unlimited speed and complete knowledge about the searchers' movements. The fugitive is caught if (1) he or she is on an edge guarded at both ends by searchers and a searcher is moved through the edge or (2) he or she is on an edge from which there is no escape and a searcher is moved through this edge. The searchers may be added to any vertex at any time, deleted from a vertex at any time, and moved at any time from one end of an edge to the other end. An edge $e$ of a graph $G$ has been *cleared* after a given sequence of searcher movements if it is not possible for the fugitive

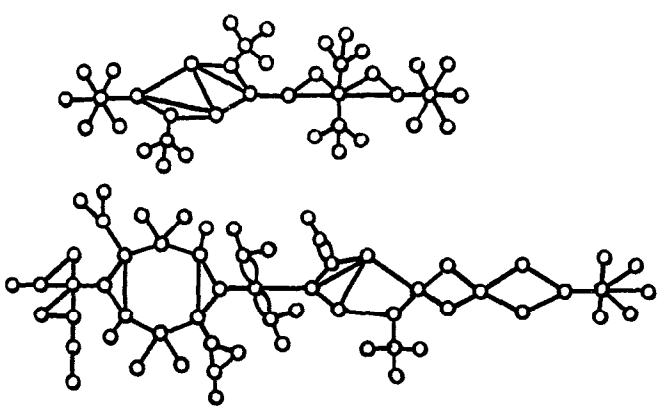

Fig. 6. Some graphs with cutwidth 3.

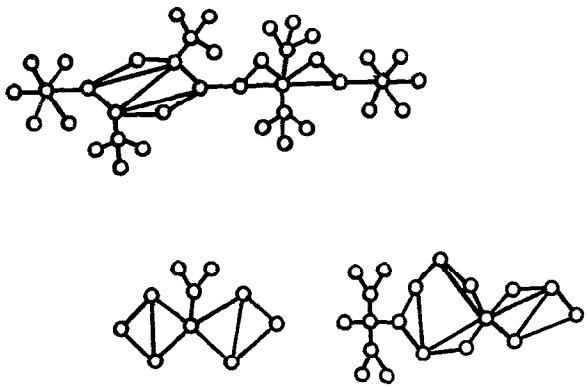

Fig. 7. Some graphs with cutwidth greater than 3.

to be on that edge without having been caught. *Recontamination* occurs to an edge $e$ if it has been cleared and, after an additional sequence of searcher movements, it is again possible for the fugitive to be on that edge. LaPaugh [7] has recently shown that, if $s(G)=k$, then there is a sequence of searcher movements in which recontamination does not occur, the fugitive is caught, and $k$ searchers are used. That is, allowing recontamination does not reduce the number of searchers needed to clear all edges of a graph.

It is known that search number and cutwidth are identical for trees having maximum vertex degree 3 [2]. It is also known that search number can be considerably smaller than cutwidth for arbitrary trees. For example, for all $k \geq 1$, the star $S_k$, i.e. the tree with $k+1$ vertices and $k$ leaves, has search number 2 and cutwidth $\lceil \frac{1}{2}k \rceil$. The following result shows that search number is never greater than cutwidth.

**Lemma 3.1.** *For any graph* $G$, $s(G) \leq cw(G)$.

**Proof.** An algorithm is presented for searching a graph in which the number of searchers is bounded by the cutwidth. Let $G=(V,E)$ be an arbitrary graph. Let $L$ be a linear layout of $G$ such that $cw(G,L)=k$. We shall say that an edge $e=\{x,y\}$ is in the *ith cut* (under the layout $L$) if $L(x) \leq i$ and $L(y) > i$. The algorithm constructed has as its "loop invariant" the property that during the $i$th iteration a searcher is located on the left end of each edge in the $i$th cut. During the $i$th iteration, searchers are moved through all edges that are in the $i$th cut and are incident to the $(i+1)$st vertex. The number of searchers on the graph is then adjusted to maintain the loop invariant. Since $cw(G,L) \leq k$, each cut has at most $k$ edges. Therefore, the algorithm needs at most $k$ searchers. The algorithm is called SEARCH and is presented below:

```
procedure SEARCH(G)
begin
  for j=1 step 1 until |G| do
    begin
      let y be the jth vertex under the layout L;
      for each vertex x such that x is assigned a position to the left of
          y under the layout L and {x,y} is an edge in G do
            move a searcher through the edge {x,y} from vertex x to
            vertex y;
      let m be the number of edges incident to y that are in the ith cut
      while there are fewer than m searchers on y do
        add a new searcher to vertex y;
      while there are more than m searchers on y do
        delete a searcher from y;
    end
end
```

The correctness of this algorithm follows from the straightforward observation that the loop invariant is maintained throughout. That is, since all edges have searchers moved through them and the fugitive is prevented from moving back into cleared edges by the positions of the searchers, the fugitive is eventually caught. There are never more than $k$ edges in any cut of the graph $G$ under the layout $L$, since $L$ is a cutwidth $k$ layout of $G$. Therefore, the algorithm never places more than $k$ searchers on $G$. □

So, the search number of a graph $G$ is never larger than its cutwidth. In fact, as we will show, the search number of a graph $G$ is identical to its cutwidth, provided that the graph has maximum vertex degree 3.

How does one show that, for any degree 3 graph $G$, $cw(G) \leq s(G)$? The most natural approach would seem to be to go from a search strategy which does not allow recontamination to a linear layout by assigning a vertex to the integer $i$ if it is the $i$th vertex to be visited by a searcher. This, however, does not work. We can construct examples of graphs with maximum vertex degree 3 in which the cutwidth of a layout obtained in this way is larger that the number of searchers used.

In fact, an appropriate layout is obtained by assigning vertices to integers in the order in which at least half of their incident edges are cleared. This is shown in the following result.

**Theorem 3.2.** *For any graph $G$, $cw(G) \leq \lfloor \frac{1}{2} \deg(G) \rfloor \cdot (s(G)-1)+1$, where $\deg(G)$ denotes the maximum degree of any vertex in $G$.*

**Proof.** Let $G$ be an arbitrary undirected, connected, and finite graph. Let $G'$ be the reduction of $G$. Let $G^{(2)}$ be the graph obtained from $G'$ by adding a single degree

two vertex into every self loop of $G'$ (if there are any). It is easily seen that $s(G)=s(G^{(2)})$ and, as we have already observed in Section 2, $cw(G)$ is identical to $cw(G^{(2)})$. Therefore, without any loss of generality, we may show the result for $G^{(2)}$. In fact, for ease of notation, we simply assume that $G=G^{(2)}$.

Let $s(G)=k$. So, there is a sequence $S$ of movements of searchers which (a) never places more than $k$ searchers on $G$ simultaneously, (b) never allows recontamination, and (c) clears all of the edges of $G$. Our goal is to describe a linear layout $L_S$ of $G$ such that $cw(G,L_S)\leq \lfloor \frac{1}{2}\deg(G)\rfloor \cdot (k-1)+1$.

Define the function $f_S$ which maps vertices of $G$ into natural numbers by: $f_S(x)=i$ if and only if $i$ is the smallest integer such that after the $i$th step of $S$ at least half of the edges incident to $x$ are cleared. The function $f_S$ is not in general a layout, since during the $i$th step an edge connecting vertices $x$ and $y$ may be cleared and both $x$ and $y$ may satisfy the stated property for the first time. So, let $L_S$ be an arbitrary layout of $G$ satisfying the following properties:

(1) if $f_S(x)<f_S(y)$, then $L_S(x)<L_S(y)$,

(2) if $f_S(x)=f_S(y)$, degree$(x)=2$, and the set $\{z \mid \{y,z\}$ is an edge and $f_S(z)<f_S(y)\}$ contains at least $\lfloor \frac{1}{2}\deg(G)\rfloor$ elements, then $L_S(y)<L_S(x)$; otherwise, $L_S(x)<L_S(y)$.

In other words, the layout $L_S$ is obtained from $f_S$ by arbitrarily deciding which of two vertices assigned to the same integer by $f_S$ gets the smaller number, except when one of the two vertices has degree two. When two vertices, say $x$ and $y$, are mapped to the same integer by $f_S$ and $x$ has degree 2, then $x$ is given the smaller integer only when $y$ is connected to fewer than $\lfloor \frac{1}{2}\deg(G)\rfloor$ vertices that have smaller values than $y$ under the mapping $f_S$.

We show that there are at most $\lfloor \frac{1}{2}\deg(G)\rfloor \cdot (k-1)+1$ edges in any one of the successive cuts of $G$ under the layout $L_S$. The bound on the number of edges in the $i$th cut, for each $i$ ($1\leq i\leq |\text{vertices}(G)|$), follows by showing, as we do, that there are at most $\lfloor \frac{1}{2}\deg(G)\rfloor$ edges in this cut for each searcher that is not moved during step $t_i$ and a single edge in this cut for the one searcher that is moved during step $t_i$, where $t_i$ is the step in the search sequence $S$ when the $i$th vertex first has at least half of its incident edges cleared. Since there are at most $k$ searchers used in the sequence $S$, there are at most $k-1$ stationary searchers at any step and, consequently, the indicated bound on the size of the $i$th cut follows.

Consider now the edges in the $i$th cut. These edges are in one of the following sets: (a) the set of edges $C(i)$ that have been cleared before step $t_i$, (b) the set of edges $N(i)$ not cleared at the end of step $t_i$, and (c) the singleton set containing the edge $e_i$ which is cleared during step $t_i$.

We show first that each edge in $C(i)$ is incident to a vertex to the right of the $i$th vertex that contains a stationary searcher during step $t_i$ and that at most $\lfloor \frac{1}{2}\deg(G)\rfloor$ edges in $C(i)$ are incident to a common vertex to the right of the $i$th vertex. Let $e=\{x,y\}$ be an edge in $C(i)$. We assume that $L_S(x)<L_S(y)$. The vertex $y$ is inci-

dent also to at least two edges that are not cleared before step $t_i$. That is, if $y$ were incident to at most one edge not cleared before step $t_i$, then at least half of its incident edges would be cleared and $y$ would be assigned to a position to the left of the $i$th vertex by the layout $L_S$. However, $y$ lies to the right of the $i$th vertex. Consequently, a stationary searcher must be located on $y$ during step $t_i$, since $y$ is incident to at least two uncleared edges and at least one cleared edge and no recontamination is allowed. Also, if any vertex to the right of the $i$th vertex were incident to more than $\lfloor \frac{1}{2}\deg(G) \rfloor$ edges in $C(i)$, then at least half of its incident edges would be cleared before step $t_i$ and so it would be positioned to the left, not the right, of the $i$th vertex.

Each edge in $N(i)$ is incident to a vertex to the left of the $(i+1)$st vertex that either (a) contains a stationary searcher during step $t_i$ or (b) is a degree two vertex that is also incident to the edge $e_i$ cleared during step $t_i$. Any vertex to the left of the $(i+1)$st vertex must be incident to at least one edge that is cleared at the end of step $t_i$, since at least half of its incident edges are cleared. Consequently, any vertices to the left of the $(i+1)$st vertex that are also incident to an uncleared edge must contain a searcher to prevent recontamination. So, if $\{x, y\}$ is an edge in $N(i)$, where $L_S(x) < L_S(y)$, then the vertex $x$ must contain a searcher at the end of step $t_i$. If the cleared edge incident to $x$ is not $e_i$, i.e. the edge cleared during step $t_i$, then $x$ contains a stationary searcher during step $t_i$. Similarly, if $x$ has degree at least three, then at least two edges incident to $x$ must be cleared at the end of step $t_i$ and, consequently, at least one edge incident to $x$ was cleared before step $t_i$. So, again in this case, the vertex $x$ contains a stationary searcher during step $t_i$. It follows that the only vertex to the left of the $(i+1)$st vertex that may be incident to an edge in $N(i)$ and not contain a stationary searcher during step $t_i$ is a degree two vertex that is also incident to $e_i$.

Furthermore, if any vertex to the left of the $(i+1)$st vertex were incident to more than $\lfloor \frac{1}{2}\deg(G) \rfloor$ edges in $N(i)$, then half of its incident edges would not be cleared at the end of step $t_i$ and this would contradict the definition of the layout $L_S$. That is, all vertices up to the $i$th vertex in the layout $L_S$ must have at least half of their incident edges cleared by the end of step $t_i$. So, each stationary searcher corresponds to at most $\lfloor \frac{1}{2}\deg(G) \rfloor$ edges in the $i$th cut.

There is still the possibility, of course, that a single edge in $N(i)$ is incident to a vertex to the left of the $(i+1)$st vertex that does not contain a stationary searcher during step $t_i$. As we have seen, this vertex must be a degree two vertex that is also incident to the edge $e_i$ cleared during step $t_i$. We must look carefully at this case, since it would seem at first glance to upset the counting arrangement for edges in the $i$th cut.

Let $e_i = \{x, y\}$ and let the searcher move from vertex $y$ to vertex $x$ during step $t_i$. As indicated above, vertex $x$ in this case has degree two and is also incident to a single edge in $N(i)$. If the edge $e_i$ is not in the $i$th cut, then the edge in $N(i)$ incident to vertex $x$ can be made to correspond to the searcher moved during step $t_i$ and the number of edges in the $i$th cut are bounded in the way specified. However, if $e_i$ and

the edge in $N(i)$ incident to $x$ are both in the $i$th cut, then we cannot make both edges correspond to the searcher moved in step $t_i$. Since degree two vertices in $G$ are only those that exist in self-loops, the edge in $N(i)$ and incident to $x$ must also connect vertex $x$ with vertex $y$. The situation is described in Fig. 8.

It follows that there must be a stationary searcher on vertex $y$ during step $t_i$, since recontamination must be prevented. There are two cases to consider: (a) the vertex $y$ has fewer than half of its incident edges cleared after step $t_i$ and (b) vertex $y$ has at least half of its incident edges cleared after step $t_i$.

In case (a), since vertex $y$ has fewer than half of its incident edges cleared after step $t_i$, there are at most $\lfloor \frac{1}{2}\deg(G) \rfloor$ cleared edges incident to $y$ including the edge $e_i$ cleared during step $t_i$. So, the edge $e_i$ can be made to correspond to the stationary searcher on vertex $y$ and the edge $\{x, y\}$ in $N(i)$ can be made to correspond to the searcher moved in step $t_i$.

In case (b), since $y$ has at least half of its incident edges cleared after step $t_i$ and yet lies to the right of the $i$th vertex, it follows that $f_S(x) = f_S(y)$. By the definition of the layout $L_S$, at most $\lfloor \frac{1}{2}\deg(G) \rfloor$ edges incident to $y$ are in the $i$th cut; otherwise, the vertex $x$ would be to the right of $y$ and not to the left of $y$. Consequently, at most $\lfloor \frac{1}{2}\deg(G) \rfloor - 1$ edges in $C(i)$ are incident to $y$. So, the edge $e_i$ can be included with those edges in $C(i)$ that correspond to the stationary searcher on vertex $y$ and the total number of edges corresponding to this searcher is bounded by $\lfloor \frac{1}{2}\deg(G) \rfloor$. The edge $\{x, y\}$ in $N(i)$ can then be made to correspond to the searcher moved during step $t_i$.

In conclusion, we have shown that for each stationary searcher during step $t_i$ there are at most $\lfloor \frac{1}{2}\deg(G) \rfloor$ edges in the $i$th cut, there is one edge in the $i$th cut corresponding to the searcher moved during step $t_i$, and every edge in the $i$th cut corresponds to a searcher. Consequently, since there are at most $k-1$ stationary searchers at step $t_i$, there are at most $\lfloor \frac{1}{2}\deg(G) \rfloor \cdot (s(G)-1)+1$ edges in the $i$th cut. Since the integer $i$ was chosen arbitrarily, the result $cw(G, L_S) \leq \lfloor \frac{1}{2}\deg(G) \rfloor \cdot (s(G)-1)+1$ follows. □

The following result follows immediately from Lemma 3.1 and Theorem 3.2.

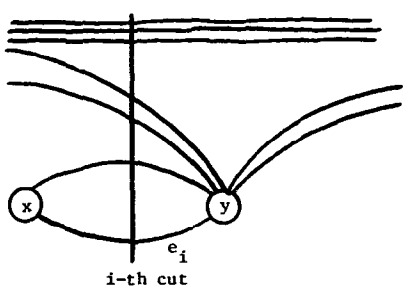


Fig. 8. A situation in which an edge in $N(i)$ is incident to a degree 2 vertex.

**Corollary 3.3.** *For any graph $G$ with maximum vertex degree three, $s(G) = cw(G)$.*

We observe that the result indicated in Theorem 3.2 is almost the best possible. That is, we have seen that, for all $k \geq 2$, the star $S_k$, which has maximum vertex degree $k$, satisfies $s(S_k) = 2$ and $cw(S_k) = \lceil \frac{1}{2}k \rceil$. Consequently, for all odd values of $k$,

$$cw(S_k) = \lceil \tfrac{1}{2}k \rceil = \lfloor \tfrac{1}{2}k \rfloor + 1 = \lfloor \tfrac{1}{2}\deg(S_k) \rfloor \cdot (s(S_k) - 1) + 1.$$

In [10] it was shown that, for any graph $G$, $s(G) \leq b(G) + 1$, where $b(G)$ denotes the bandwidth of the graph $G$. (The *bandwidth of a graph $G$ under a linear layout $L$* is $\max\{|L(x) - L(y)| \mid \{x, y\} \text{ is an edge in } G\}$. It is denoted by $b(G, L)$. The *bandwidth of $G$* is $\min\{b(G, L) \mid L \text{ is a linear layout of } G\}$.) Combining this with Theorem 3.2, we have the following result.

**Corollary 3.4.** *For any graph $G$, $cw(G) \leq \lfloor \frac{1}{2}\deg(G) \rfloor \cdot b(G) + 1$, where* $\deg(G)$ *denotes the maximum degree of any vertex in $G$.*

The result indicated in Corollary 3.4 is also almost the best possible. That is, for each $d, m, n \geq 1$, there is a graph $R^d_{m,n}$ such that, for sufficiently large $n$ and all $m \geq d$,

(1) $\deg(R^d_{m,n}) = 2d + 1$,
(2) $b(R^d_{m,n}) = m$, and
(3) $cw(R^1_{m,n}) = m + 1$ and, for all $d > 1$, $cw(R^d_{m,n}) = d \cdot m$.

Thus, $cw(R^d_{m,n}) \geq \lfloor \frac{1}{2}\deg(R^d_{m,n}) \rfloor \cdot b(R^d_{m,n})$.

Let $R^d_{m,n}$ be the $m \times n$ rectangle graph with vertex degree $2d + 1$, whose vertices are those in the set $\{(i, j) \mid 1 \leq i \leq m,\ 1 \leq j \leq n\}$ and whose edges are those in the following sets:

(1) $\{\{(j, i), (k, i+1)\} \mid j - d + 1 \leq k \leq j\}$ for all $i$ $(1 \leq i < n)$,
(2) $\{\{(j, i), (k, i)\} \mid j < d,\ k > m - d + 1\}$ for all $i$ $(1 \leq i \leq n)$,
(3) $\{(2j - 1, 2i - 1), (2j, 2i - 1)\}$ for all $i$ $(1 \leq i \leq \lfloor \frac{1}{2}n \rfloor)$ and all $j$ $(1 \leq j \leq \lfloor \frac{1}{2}m \rfloor)$,
(4) $\{(2j, 2i), (2j + 1, 2i)\}$ for all $i$ $(1 \leq i \leq \lfloor \frac{1}{2}n \rfloor)$ and all $j$ $(1 \leq j \leq \lfloor \frac{1}{2}m \rfloor)$,
(5) $\{\{(j, 1), (j + 1, 1)\} \mid 1 \leq j < m\}$, and
(6) $\{\{(j, n), (j + 1, n) \mid 1 \leq j < m\}$.

For example, the rectangle graph $R^2_{5,8}$ is shown in Fig. 9.

That $b(R^d_{m,n}) = m$, for sufficiently large $n$, follows from the fact that $R^d_{m,n}$ can be laid out column-by-column with bandwidth $m$. That is, all edges in this graph connect vertices in adjacent columns or in the same column and there are $m$ vertices per column. (The bandwidth cannot be smaller than $m$ without contradicting the result of Corollary 3.4, since $cw(R^d_{m,n}) = d \cdot m$, as we shall show.) The fact that $cw(R^1_{m,n}) = m + 1$ was shown in [10]. A similar argument suffices to show that $cw(R^d_{m,n}) = d \cdot m$. That is, a column-by-column layout has cutwidth $d \cdot m$ and an

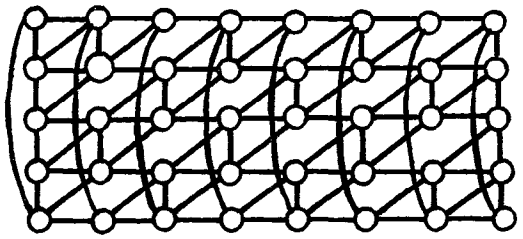

Fig. 9. The rectangle graph $R^2_{5,8}$.

argument based on the number of edge disjoint paths connecting sets of vertices shows that $cw(R^d_{m,n}) \geq d \cdot m$.

## 4. An improved dynamic programming algorithm

In [6] Gurari and Sudborough described an algorithm which, for an arbitrary graph $G$, and for any fixed value $k$, decides whether $cw(G) \leq k$ in $O(n^k)$ steps, where $n$ is the number of vertices in $G$. In this section it is shown that the earlier algorithm can be improved to yield an $O(n^{k-1})$ step process for determining if $cw(G) \leq k$, for each $k \geq 2$. Thus, for example, there is a linear time algorithm for testing cutwidth 2. A linear time algorithm has been described earlier by Garey, Graham, Johnson and Knuth [4] for testing bandwidth 2.

The basic idea in the improvement is to consider reduced graphs. We show that when the graph $G$ is reduced, except that degree two vertices are reinserted into any resulting self loops, one only needs to consider partial layouts with at most $k-1$ "active" vertices.

A *partial layout* of a graph $G=(V,E)$ is a one-to-one function $L$ mapping some subset $V'$ of the set of vertices $V$ to the set of positive integers $\{1, \ldots, |V'|\}$. The *cutwidth of the partial layout* $L$, denoted by $cw(G,L)$, is $\max\{|\mathrm{CUT}_L(i)| \mid 1 \leq i < |V'|\}$, where $\mathrm{CUT}_L(i)$ is the set of all edges in the $i$th cut (an edge $e=(x,y)$ is in the $i$th cut in this context, when $L(x)$ is defined and $L(y)$ is not defined or when $L(x) \leq i$ and $L(y) > i$). An edge $e=(x,y)$ is *dangling from the partial layout* $L$, or simply *dangling* when the partial layout is understood, if $L(x)$ is defined and $L(y)$ is not defined. A vertex $x$ is *active in the partial layout* $L$, or simply *active* when the partial layout is understood, if $x$ is incident to one of the dangling edges of $L$. The set of dangling edges and the set of active vertices of a partial layout $L$ are denoted by dangling($L$) and active($L$), respectively.

Two partial layouts $L_1$ and $L_2$ are *cutwidth equivalent* if:

(a) (active($L_1$),dangling($L_1$)) = (active($L_2$),dangling($L_2$)), and
(b) $cw(G,L_1) = cw(G,L_2)$.

In [6] Gurari and Sudborough showed that, if $L_1$ and $L_2$ are cutwidth equivalent partial layouts of a graph $G$, then $L_1$ and $L_2$ must be defined on the same set of

vertices of $G$ and either both $L_1$ and $L_2$ can be extended to cutwidth $k$ layouts or the entire graph $G$ or neither can be extended to such a complete cutwidth $k$ layout. The dynamic programming algorithm works with equivalence classes of partial layouts defined by this equivalence relation.

Clearly, no partial layout $L$ can be extended to a complete cutwidth $k$ layout if dangling($L$) contains more than $k$ edges. In fact, we show that it is not necessary to consider partial layouts with more than $k-1$ active vertices. Clearly, a partial layout with $k$ active vertices can be produced only by adding another vertex to a partial layout with $k-1$ active vertices. So, let $L$ be any partial layout with $k-1$ active vertices and at most $k$ dangling edges. There are two cases: (1) $L$ has $k$ dangling edges and (2) $L$ has $k-1$ dangling edges.

*Case* 1: *$L$ has $k$ dangling edges.* Whatever vertex is assigned next must be incident to one of the dangling edges, since there are no isolated vertices ($G$ is connected) and the number of dangling edges can never be greater than $k$. If the next vertex assigned has degree one, then the next partial layout has $k-1$ dangling edges and then either there are fewer than $k-1$ active vertices or case 2 applies. If the next vertex assigned has degree at least three or it has degree two and hence has two parallel edges connecting it to another vertex, then at least two of its edges are dangling from $L$; otherwise, the next partial layout would have more than $k$ dangling edges. So, the number of active vertices does not increase. That is, at least one vertex that was active in $L$ is no longer active. Thus, in this case, we do not produce a partial layout with more than $k$ active vertices.

*Case* 2: *$L$ has $k-1$ dangling edges.* In this case, each active vertex, is incident to exactly one of the dangling edges. If the next vertex is incident to one of these dangling edges, then the number of active vertices does not increase by adding this vertex. So, assume the next vertex $x$ is incident to one new edge. Assigning $x$ produces a partial layout $L'$ with $k$ active vertices. However, this partial layout need not be explicitly considered. That is, consider the next vertex, say $y$, that is assigned after vertex $x$. If $y$ has one incident edge, which must be one of the dangling edges from $L'$ or there would be more than $k$ dangling edges after assigning $y$, then we can switch the positions of $x$ and $y$. That is, $x, y$ cannot be an edge, since $x$ and $y$ both have degree one and the graph is connected. So, $y$ is connected to one of the vertices to the left of $x$ and, consequently, the layout with the positions of $x$ and $y$ switched does not have larger cutwidth. So, $L'$ need not be considered. The next vertex $y$ cannot have degree 2, since it must be incident to one of the dangling edges and degree 2 vertices are only incident to parallel edges.

If $y$ has degree 3, then $y$ must be incident to at least two of the dangling edges. If the dangling edges incident to $y$ do not include the one incident to $x$, then the position of $x$ and $y$ can be switched without increasing the cutwidth, as before. Again, this means that the partial layout $L'$ need not be considered. If, on the other hand, one of the dangling edges incident to $y$ is incident to $x$, then the number of active vertices in the partial layout $L''$, when $y$ is assigned, is at most $k-1$. That is, at least two formerly active vertices now become inactive. So, one simply proceeds directly

from the partial layout $L$ with $k-1$ active vertices and $k-1$ dangling edges to this new partial layout $L''$. $L''$ has at most $k-1$ active vertices. Moreover, there is not much choice for the vertices $x$ and $y$. The vertex $y$ must be incident to one of the dangling edges of the original partial layout $L$ and the vertex $x$ must be connected to $y$. Thus, only a constant number of successive partial layouts $L''$ need be considered.

A pair $p=(A,D)$, consisting of a set of vertices $A$ and a set of edges $D$, is called a *cutwidth-k-plausible* pair, if $A$ contains at most $k-1$ vertices and $D$ contains at

```
procedure CUTWIDTH_k(G)

/ A dynamic programming algorithm to test whether a graph G has cutwidth at most k. It is assumed
that the input graph G is reduced, except that a degree 2 vertex has been re-inserted into any resulting
self loops. /
[1]   begin add the pair (∅,∅) to Q;
[2]     while Q is not empty do
[3]       begin delete a pair p=(A,D) from Q;
[4]         if A is a set of k-1 vertices
[5]         then
[6]           begin
[7]             for all unassigned vertices z incident to an edge in D do
[8]               begin p'←Successor(p,z);   / let p'=(A',D') /
[9]                 if D'=∅ then stop and return "G has cutwidth at most k";
[10]                if p' is cutwidth-k-plausible and T[p'] is false then
[11]                  begin T[p']←true; add p' to Q end
[12]                for all unassigned vertices w adjacent to z do
[13]                  begin
                        p''←Successor(p,w);
[14]                    p'←Successor(p'',z);   / let p'=(A',D') /
[15]                    if D'=∅ then stop and return "G has cutwidth at most k";
[16]                    if p' is cutwidth-k-plausible and T[p'] is false then
[17]                      begin T[p']←true; add p' to Q end
[18]                  end
[19]              end
[20]          end
[21]        else   / in this case A contains fewer than k-1 vertices /
[22]          begin V←Unassigned(A,D);
[23]            for all vertices s in V do
[24]              begin p'←Successor(p,s);   / let p'=(A',D') /
[25]                if D'=∅ then stop and return "G has cutwidth at most k"
[26]                if p' is cutwidth-k-plausible and T[p'] is false then
[27]                  begin T[p']←true; add p' to Q end
[28]              end
[29]          end
[30]      end
[31]    stop and return "G has cutwidth larger than k"
[32]  end
```

Fig. 10. An algorithm to decide if a given graph $G$ has cutwidth at most $k$.

most $k$ edges. The algorithm uses two data structures: (1) a queue $Q$ whose elements are cutwidth-$k$-plausible pairs and (2) an array $T$ with one entry for each cutwidth-$k$-plausible pair. The element $T[p]$ is *true* iff the pair $p=(A,D)$ represents an equivalence class of partial layouts that has already been considered before; otherwise, $T[p]$ is *false*. At the start of the algorithm $T[p]$ is *false* for all pairs $p$.

Our improved algorithm also uses two separate procedures that were described by Gurari and Sudborough [6]. We include these procedures here for completeness. They are given in Figs. 11 and 12. Unassigned$(A,D)$ computes the set of vertices that are unassigned in any partial layout $L$ such that (active$(L)$,dangling$(L)$)$=(A,D)$. This procedure requires at most $O(n)$ steps, where $n$ is the number of vertices in the graph. Successor$(p,z)$ computes, for a given pair $p=(A,D)$ and a given vertex $z$, the pair $p'=(A',D')$ that denotes the equivalence class containing all partial layouts that result from those in the equivalence class denoted by the pair $p=(A,D)$ when the vertex $z$ is assigned. That is, $A'$ is the new set of active vertices and $D'$ is the new set of dangling edges after the vertex $z$ is assigned to the next available integer. The procedure Successor$(p,z)$ works in O(1), i.e. constant, time.

```
procedure UNASSIGNED(A,D)
begin
  V←∅;  / V is a set variable that will contain the set of unassigned vertices /
  Q'←∅;  / Q' is a queue whose elements are certain edges of the given graph /
  if D=∅ then return {x | x is a vertex of G};
  for all edges e in D do
    begin
      DG[e]←true;  / the boolean element DG[e] is true if and only if the edge e has been added
                     to the queue Q' /
      add e to Q'
    end
  for all vertices v in A do
    P[v]←true;  / the boolean element P[v] is true if and only if the vertex has been placed in V
                  or does not belong in V /
  while Q' is not empty do
    begin
      delete an edge e={x,y} from Q';
      if P[v]=false or P[y]= false then
        begin
          if P[x]=false then z←x else z←y;
          P[z]←true;
          V←V∪{z};
          for each edge e incident to z do
            if DG[e]=false
            then begin add e to Q'; DG[e]←true end
        end
    end
  return V
end
```

Fig. 11. The procedure UNASSIGNED.

```
procedure SUCCESSOR(p,s)

begin  / let p be the pair (A,D) /
  for each edge e in the set D do
    if e is incident to s then delete e from D;
  for each edge e incident to s do
    if e is not incident to a vertex in A then add e to D;
  for each vertex v in the set A do
    if v is not incident to any edge in D then delete v from A;
  if s is incident to an edge in D then add s to A;
  return the pair p' = (A,D);
end
```

Fig. 12. The procedure SUCCESSOR.

The improved algorithm is described in Fig. 10. It begins, as shown in line [1], by placing the pair $(\emptyset,\emptyset)$ in the queue $Q$. In general, while the queue $Q$ is not empty, the procedure works by taking off a pair $p=(A,D)$ from the queue $Q$ and then dividing its computation according to whether the set of active vertices $A$ contains $k-1$ elements or some smaller number of vertices.

If $A$ contains $k-1$ vertices, as shown in line [4], then the procedure tries to extend the class of partial layouts in two separate ways. The first is by trying all possible vertices that are incident to one of the dangling edges in $D$ as the next vertex in an extended partial layout, as shown in lines [8]–[11]. The second is by trying pairs of vertices, $w$ and $y$, in that order, for the next two vertices in an extended partial layout, where $y$ is incident to one of the dangling edges in $D$ and $w$ is a neighboring vertex to $y$ that has not yet been assigned, as shown in lines [12]–[17]. These are the only types of extended partial layouts that need to be considered, as we have seen already in our earlier discussion.

If $A$ contains less than $k-1$ vertices, then the procedure tries all unassigned vertices as the next vertex in an extended partial layout. That is, it looks at all equivalence classes corresponding to a pair Successor$(p,s)$, where $p$ is the current pair taken from the queue $Q$ and $s$ is an unassigned vertex.

In each of these cases the strategy is the same. The procedure determines first if the new partial layout is, in fact, a complete layout of the given graph. This is true if and only if the set of dangling edges in the new class of layouts is empty, as shown in lines [9], [15], and [25] of the procedure. If there is a complete layout of the given graph that is obtained from a cutwidth-$k$-plausible pair in the queue $Q$, then it follows easily that this layout has cutwidth at most $k$ and, consequently, the procedure returns the appropriate answer and terminates. On the other hand, if the new partial layout is not a complete layout of the given graph, then a decision is made about adding the corresponding new pair $p'=(A',D')$ to the queue $Q$. It is added to the queue if and only if it has not been considered before, i.e. $T[p']$ is false, and it is cutwidth-$k$-plausible, as shown in lines [10], [16], and [26]. It follows, of course, that the only pairs added to the queue $Q$ are cutwidth-$k$-plausible pairs and each such cutwidth-$k$-plausible pair is added at most one time to the queue $Q$. (The last

observation follows from the fact that we set the value $T[p']$ to true when $p'$ is added to the queue $Q$.)

The correctness of the algorithm follows from the earlier discussion that shows that the search for a cutwidth $k$ layout of the entire given graph can be limited to the type of partial layouts with $k-1$ active vertices and that shows that the only partial layouts that need to be considered to extend a partial layout with $k-1$ active vertices are, in fact, those that the procedure considers.

We turn now to the analysis of the running time of this procedure. Observe first that there are $O(n^{k-1})$ pairs of the form $p=(A,D)$ with $A$ a set of $k-1$ vertices and that there are $O(n^{k-2})$ pairs of the form $p=(A,D)$ with $A$ a set of less than $k-1$ vertices. For each of the pairs $p=(A,D)$ that is cutwidth-$k$-plausible, there is the possibility that the main loop of the procedure, namely lines [2]-[30], will be executed. Furthermore, in each such execution of the main loop either the lines [6]-[20] or the lines [22]-[29] will be executed, depending upon whether there are $k-1$ active vertices or less, respectively. Observe that each execution of lines [6]-[20] takes $O(1)$ steps, since there are at most $k$ vertices incident to edges in $D$ and, for each such vertex, there are at most $2k$ neighbors. Also each execution of lines [22]-[29] requires at most $O(n)$ steps, since there may be $O(n)$ vertices that are yet unassigned and the procedure UNASSIGNED requires at most $O(n)$ steps to enumerate them. Consequently, the total number of steps is bounded by the number of times that lines [6]-[20] can be executed times the number of steps needed to execute these lines plus the number of times lines [22]-[29] can be executed times the number of steps needed to execute these lines. As we have seen, this is bounded by $O(n^{k-1})\cdot O(1)+O(n^{k-2})\cdot O(n)$. Consequently, the number of steps needed to execute the complete procedure is bounded by a function in $O(n^{k-1})$. This result is expressed in the following theorem.

**Theorem 4.1.** *For each $k\geq 2$, the problem of deciding, for a given graph $G$, if $cw(G)\leq k$ or not can be solved in $O(n^{k-1})$ steps.*

Clearly, as a corollary, we have that graphs with cutwidth 2 can be recognized in linear time. Although the result does not follow from the approach given here, we recall that the results of Section 2 strongly suggest that there is a linear time algorithm to recognize graphs with cutwidth three as well.

## References

[1] M.A. Breuer, Min cut placement, J. Design Automation and Fault-Tolerant Comput. 1 (1977) 343-362.

[2] M.-J. Chung, F. Makedon, I.H. Sudborough and J. Turner, Polynomial algorithms for the min-cut linear arrangement problem on degree restricted trees, SIAM J. Comput. 14 (1985) 158-177.

[3] A. Feller, Automatic layout of low-cost quick turnaround random-logic custom LSI devices, in: Proceedings 13th Design Automation Conference (1976) 79-85.

[4] M.R. Garey, R.L. Graham, D.S. Johnson and D.E. Knuth, Complexity results for bandwidth minimization, SIAM J. Appl. Math. 34 (1978) 477–495.

[5] F. Gavril, Some NP-complete problems on graphs, in: Proceedings 11th Annual Conference on Information Sciences and Systems, Baltimore, MD (1977) 91–95.

[6] E.M. Gurari and I.H. Sudborough, Improved dynamic programming algorithms for bandwidth minimization and the min cut linear arrangement problem, J. Algorithms 5 (1984) 531–546.

[7] A.S. LaPaugh, Recontamination does not help to search a graph, Tech. Rept., Electrical Engineering and Computer Science Department, Princeton University, Princeton, NJ (1983).

[8] T. Lengauer, Upper and lower bounds on the complexity of the min cut linear arrangement problem on trees, Tech. Rept. TM-80-1272-9, Bell Laboratories, Murray Hill, NJ (1980).

[9] A.D. Lopez and H.-F.S. Law, A dense gate matrix layout method for MOS VLSI, IEEE Trans. Electron. Devices 27 (8) (1980) 1671–1675.

[10] F. Makedon, C.H. Papadimitriou and I.H. Sudborough, Topological bandwidth, SIAM J. Algebraic Discrete Methods 6 (1985) 418–444.

[11] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson and C.H. Papadimitriou, The complexity of searching a graph, in: Proceedings IEEE Foundations of Computer Science Symposium (1981) 376–385.

[12] T. Ohtsuki, H. Mori, E.S. Kuh, T. Kashiwabara and T. Fujisawa, One-dimensional logic gate assignments and interval graphs, IEEE Trans. Circuits Syst. 26 (9) (1977) 675–684.

[13] T.D. Parsons, The search number of a connected graph, in: Proceedings 9th Southeastern Conference on Combinatorics, Graph Theory and Computing (1978) 549–554.

[14] T.D. Parsons, Pursuit evasion in a graph, in: Y. Alavi and D.R. Lick, eds., Theory and Application of Graphs (Springer, Berlin, 1976) 426–441.

[15] G. Persky, D. Deutsch and D. Schweikert, LTX: A minicomputer-based system for automated LSI layout, J. Design Automation and Fault Tolerant Comput. 1 (1977) 217–255.

[16] L. Stockmeyer, Personal communication to M. Garey and D.S. Johnson (1974), in: M. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, San Francisco, CA, 1979).

[17] S. Trimberg, Automating chip layout, IEEE Spectrum (1982) 38–45.

[18] A. Weinberger, Large scale integration of MOS complex logic: A layout method, IEEE J. Solid State Circuits 2 (1967) 182–190.

[19] M. Yannakakis, A polynomial algorithm for the min cut linear arrangement of trees, J. ACM 32 (4) (1988) 950–959.

[20] H. Yoshizawa, H. Kawanishi and K. Kani, A heuristic procedure for ordering MOS arrays, in: Design Automation Conference (1975) 384–393.