JOURNAL OF ALGORITHMS **14**, 1–23 (1993)

On Linear Time Minor Tests with Depth-First Search

HANS L. BODLAENDER

Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

Received May 28, 1991; revised August 1991

Recent results on graph minors make it desirable to have efficient algorithms that, for a fixed set of graphs $\{H_1, \dots, H_c\}$, test whether a given graph G contains at least one graph H_i as a minor. In this paper we show the following result: if at least one graph H_i is a minor of a $2 \times k$ grid graph, and at least one graph H_i is a minor of a circus graph, then one can test in $\mathcal{O}(n)$ time whether a given graph G contains at least one graph $H \in \{H_1, \dots, H_c\}$ as a minor. This result generalizes a result of Fellows and Langston. The algorithm is based on depth-first search and on dynamic programming on graphs with bounded treewidth. As a corollary, it follows that the MAXIMUM LEAF SPANNING TREE problem can be solved in linear time for fixed k. We also discuss that, with small modifications, an algorithm of Fellows and Langston can be modified to an algorithm that finds in $\mathcal{O}(k!\,2^k n)$ time a cycle (or path) in a given graph with length $\geq k$ if it exists. © 1993 Academic Press, Inc.

1. INTRODUCTION

A graph H is a minor of a graph G if H can be obtained from G by a series of edge deletions and edge contractions. An edge contraction is the operation where two adjacent vertices v, w are replaced by a new vertex that is adjacent to each vertex that was adjacent to v or w. Recent results of Robertson and Seymour [25, 27] provide us with a powerful tool to determine the complexity of deciding whether a given graph G belongs to some class of graphs that is closed under taking of minors. Let A be a class of graphs that is closed under taking of minors, i.e., if $G \in A$ and H is a minor of G, then $H \in A$. Robertson and Seymour proved that for each such class of graphs, there exists a *finite* set of graphs ob(A), the obstruction set of A, such that for all graphs G: $G \in A$, if and only if there is no $H \in$ ob(A) that is a minor of G [25]. As one can test for fixed graphs H in $\mathcal{O}(n^3)$ time whether a given graph G contains H as a minor [27], this shows the existence of a method to test in $\mathcal{O}(n^3)$ membership in minor closed classes of graphs. Many applications of these results were found by Fellows and Langston [12–14].

Using the notions of treewidth and tree decomposition, Robertson and Seymour showed that if A does not contain all planar graphs, then $\mathcal{O}(n^2)$ algorithms exist. This is done by showing the following three results: first, for every planar graph H, there exists a constant c_H, such that for all graphs G that do not contain H as a minor, the treewidth of G is at most c_H. Second, for every fixed k, they showed that there exists an $O(n^2)$ algorithm, that given a graph G either decides that the treewidth of G is more than k, or it finds a tree decomposition of G with treewidth $O(k)$. Finally, when a graph G is given, together with a tree decomposition of G with constant bounded treewidth, then all minor tests on G can be done in linear time. A very serious drawback of this approach is that the constant factor is enormously large (making the algorithms totally unpractical). (See [20] for a discussion of these results.) Very recently, Lagergren showed that the second step can be done in $\mathcal{O}(n \log^2 n)$ time [21]. This shows that minor closed classes of graphs that do not contain all planar graphs have an $\mathcal{O}(n \log^2 n)$ membership test algorithm.

Fellows and Langston showed that if A avoids at least one cycle (or minor of a cycle) then there exist $\mathcal{O}(n)$ algorithms [17]. The basic idea here is that there is an $O(n)$ algorithm, that either directly finds the cycle, or finds a tree decomposition of the input graph with constant bounded treewidth. Given this tree decomposition, all minor tests can then be done in linear time. In this paper we extend the results of Fellows and Langston. We show that if A avoids at least one $2 \times k$ grid graph and at least one circus-graph (see Section 2 for definitions), then one can test membership in A in linear time. As a corollary it follows that for fixed k, one can test in linear time whether a given graph $G = (V, E)$, contains a spanning tree with at least k leaves. Our results are optimal in the (limited) sense that no other minor-closed classes can be dealt with using the same technique. It should be noted that our results do not have the very large constants of Robertson and Seymour, and that the algorithms may even be practical in some cases.

Note that the graph minor theorem of Robertson and Seymour is nonconstructive. I.e., in order to *write down* the algorithms, one must know the obstruction set. Thus, we can end up in the situation where we know that an $\mathcal{O}(n)$, $\mathcal{O}(n \log^2 n)$, or $\mathcal{O}(n^3)$ algorithm exists, but we do not know how to construct it. However, in many important cases, one can avoid the nonconstructive elements of the graph minor theorem by using self-reductions [5, 7, 17]. Also, in several important cases, one can compute the obstruction sets [16].

Recently, Arnborg *et al.* [1] showed that every class of graphs definable in monadic second-order form can be recognized in linear time (under the uniform cost measure) by an algorithm based on graph reduction. (The algorithm may read memory locations that never received a value written to it. Each access to a memory locations is assumed to take constant time.) The algorithm uses polynomial (not linear) space. As minor-tests are expressible in monadic second order form [8], one has as a corollary that every minor closed class A that does not contain all planar graphs can be recognized in $\mathcal{O}(n)$ time (under the uniform cost measure). Note, however, that this approach still suffers from the very large constants associated with the results of Robertson and Seymour, cited above. In contrast, our paper has much better constant factors and has a more direct approach, but can deal only with a much smaller collection of graph classes.

This paper is organized as follows. In Section 2 we give a number of definitions and preliminary results. In Section 3 we give our main construction. In Section 4 we show the consequences of the results of Section 3. A number of final comments are made in Section 5.

2. Definitions and Preliminary Results

In this section we give some preliminary definitions and results. The notion of tree decomposition was introduced by Robertson and Seymour [26].

Definition. Let $G = (V, E)$ be a graph. A tree decomposition of G is a pair $(\{X_i | i \in I\}, T = (I, F))$ with $\{X_i | i \in I\}$ a family of subsets of V, and T a tree, with the following properties:

- $\bigcup_{i \in I} X_i = V$
- For every edge $e = (v, w) \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$.
- For every $v \in V$, the set $\{i | v \in X_i\}$ forms a connected subtree of T.

The treewidth of a tree decomposition $(\{X_i | i \in I\}, T)$ is $\max_{i \in I} |X_i| - 1$. The treewidth of G, denoted by treewidth(G), is the minimum treewidth of a tree decomposition of G, taken over all possible tree decompositions of G.

There are several alternative ways to characterize the class of graphs with treewidth $\geq k$, e.g., as partial k-trees. (See, e.g., [28, 29].) A large number of NP-complete (and other) graph problems can be solved in polynomial and even linear time, when restricted to graphs with constant treewidth (see, e.g., [2–4, 9, 22, 28, 30]). We mention two specific results:

THEOREM 2.1. *Let k be a constant. Let H be a fixed graph. There exists a linear time algorithm, that given a graph G with a tree decomposition of G with treewidth $\leq k$, decides whether G contains H as a minor.*

The proof of Theorem 2.1. can be found in [8 or 27].

THEOREM 2.2. *There exists an algorithm, that uses $\mathcal{O}(2^k k! n)$ time, and finds the longest cycle (or longest path) in a given graph G, that is given together with a tree decomposition of G with treewidth $\leq k$.*

Proof. For constant k, this problem can be solved in $\mathcal{O}(n)$ time with dynamic programming, as in [3, 4, 6, or 30]. By paying careful attention to a number of not very interesting implementation details, one can achieve a constant factor of $c \cdot 2^k k!$, c not depending on k. □

In this paper we use a special type of tree decompositions: those that are "based on a spanning tree" T of G.

DEFINITION. Let $T = (V, F)$ by a spanning tree of graph G. A T-based tree decomposition of G is a tree decomposition $(\{X_v | v \in V\}, T = (V, F))$, with for all $v \in V : v \in X_v$.

LEMMA 2.3. *Let $(\{X_v | v \in V\}, T = (V, F))$ be a T-based tree decomposition of G; T is a spanning tree of G. Then for every edge $(x, y) \in E$, that is not in the spanning tree T, and for every v that is on the path from x to y in T: $x \in X_v$ or $y \in X_v$.*

Proof. Suppose not. Then consider a vertex v on a path from x to y with $(x, y) \in V$, $x \notin X_v$, $y \notin X_v$. Then as $x \in X_x$, $y \in X_y$, and $\{w | x \in X_w\}$ and $\{w | y \in X_w\}$ form connected subtrees of T, there cannot be a vertex w with $x \in X_w$ and $y \in X_w$. Contradiction. □

LEMMA 2.4. *Let $G = (V, E)$ be a graph with treewidth $\leq k$. Then $|E| \leq k \cdot |V| - \frac{1}{2}k(k + 1)$.*

Proof. This follows directly from the fact that every graph with treewidth $\leq k$ is a subgraph of a k-tree see, e.g., [28, 29]. A k-tree with n vertices has exactly $kn - \frac{1}{2}k(k + 1)$ edges. The latter fact can easily be proved with introduction to n. □

Next we give the definition of the $n \times m$ grid graphs and of the circus graphs.

DEFINITION. 1. The $n \times m$ grid graph is the graph $GR_{n \times m} = (V_{n \times m}, E_{n \times m})$, with $V_{n \times m} = \{(i, j) | 0 \leq i \leq n - 1,\ 0 \leq j \leq m - 1\}$ and $E_{n \times m} = \{((i_1, j_1), (i_2, j_2)) | (i_1, j_1), (i_2, j_2) \in V_{n \times m}$, and $((|i_1 - i_2| = 1 \wedge j_1 = j_2)$ or $(i_1 = j_2 \wedge |j_1 - j_2| = 1))\}$.

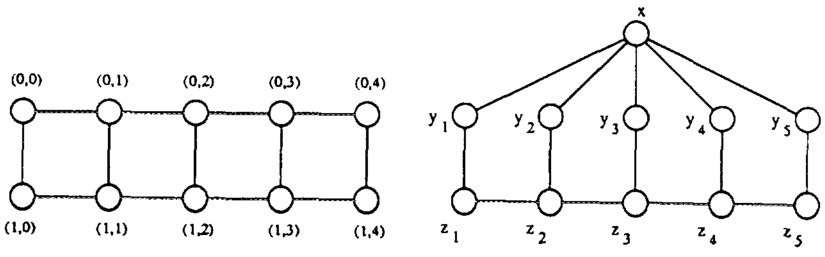

FIG. 1. $GR_{2\times 5}$ and CC_5.

2. The lth circus graph is the graph $CC_l = (\{x\} \cup \{y_i | 1 \le i \le l\} \cup \{z_i | 1 \le i \le l\},\ \{(x, y_i) | 1 \le i \le l\} \cup \{(y_i, z_i) | 1 \le i \le l\} \cup \{(z_i, z_{i+1}) | 1 \le i \le l - 1\})$.

The 2×5 grid graph and the fifth circus graph are shown, as examples, in Fig. 1. A *matching* in a graph $G = (V, E)$ is a set of edges $F \subseteq E$, such that no two edges in F share an endpoint. We assume the reader to be familiar with depth-first search and related notions.

Given a connected graph $G = (V, E)$, a depth-first search spanning tree $T = (V, F)$ of G, and a vertex $v \in V$, an edge (w, x) is said to be *around* v, if and only if $(w, x) \notin F$ is a nontree edge between a predecessor of v, and v or a descendant of v. (Note that v lies on the path between w and x in T.)

The following result of Erdös and Szekeres is needed for the proof of Lemma 3.1. A subsequence of a sequence of numbers $a_1, \dots, a_c$ is a sequence $a_{s(1)}, \dots, a_{s(c')}$ with $1 \le s(i) < s(i + 1) \le c$ for $i = 1, \dots, c' - 1$. A sequence $a_1, \dots, a_c$ is increasing if for all i, $1 \le i \le c - 1$: $a_i < a_{i+1}$. It is decreasing if for all i, $1 \le i \le c - 1$: $a_i > a_{i+1}$.

THEOREM 2.5 (Erdös and Szekeres [10]). *Let $a_1, \dots, a_c$ be a sequence of c different numbers, i.e., $1 \le i < j \le c \Rightarrow a_i \ne a_j$. If $c \ge (k - 1)^2 + 1$, then the sequence $a_1, \dots, a_c$ has an increasing subsequence of length $\ge k$, or it has a decreasing subsequence of length $\ge k$.*

3. MAIN ALGORITHM AND ITS ANALYSIS

In this section we give the main algorithm and its analysis. The following lemma plays a fundamental role in our construction.

LEMMA 3.1. *Let $G = (V, E)$ be a connected graph, and let $T = (V, F)$ be a depth-first search spanning tree of G. Let $v \in V$. Consider the subgraph of G, consisting of all nontree edges around v. If this subgraph contains a* *matching with at least* $(k-1)^2(l-1)+1$ *edges, then* G *contains a* $2 \times k$ *grid graph as a minor or* G *contains the* lth *circus graph as a minor.*

Proof. Let $G = (V, E)$, $T = (V, F)$, v be given. Suppose the subgraph of G, consisting of the edges around v contains a matching with at least $(k-1)^2(l-1)+1$ edges. Let E' be the set of edges in this matching. Let $W \subseteq V$ be the set of vertices, that are a successor of v or v itself, that are endpoint of an edge in E'. Let W' be the set of vertices $w \in W$ that have no successor w' of w with $w' \in W$. Now for every $x \in W$: either $x \in W'$, or x is on a path from v to a vertex $w \in W'$. By a pigeon-hole argument it follows that at least one of the following two cases must hold:

1. $|W'| \geq l$.
2. There exists a vertex $w \in W'$, such that at least $(k-1)^2+1$ vertices from W are on the path from v to w (v, w inclusive).

First we consider the case that $|W'| \geq l$. We claim that G contains the lth circus graph as a minor. The case that $l = 1$ is trivial. So suppose that $l \geq 2$. It follows that $v \notin W'$. One can see that G contains CC_l as a minor as follows:

Remove all nontree edges from G that are not in E'. Remove all nontree edges from G that do not have an endpoint in W'. Remove all vertices (and adjacent edges), that are not a predecessor of v, a successor of v, or v itself. Remove all vertices (and adjacent edges) that are a descendent of a vertex $w \in W'$. Remove all vertices, that are not in W' or a predecessor of a vertex in W'. Remove the edge from v to its father. Repeat the following two operations, until they are not possible anymore:

- Let w be a child of v and $w \notin W'$. Contract v and w. Let v be the name of the new vertex.
- Let w be a predecessor of v, or a vertex that is obtained by contracting a number of predecessors of v, such that w is not adjacent to a vertex in W' (in the current graph). Contract w with one of its neighbors.

Now, we end up with a graph, isomorphic to the $|W'|$th circus graph. It contains the lth circus graph as a minor, as $|W'| \geq l$. In Fig. 2 we illustrate the construction.

Next we consider the case that there exists a vertex $w \in W'$, such that the path from v to w (v, w inclusive) contains at least $(k-1)^2+1$ vertices from W. We now claim that G contains the $2 \times k$ grid graph as a minor. Let w be given. Let W'' be the set of vertices in W that are on the path from v to w. $|W''| \geq (k-1)^2+1$. Let E'' be the set of edges in E', that have an endpoint in W''. $|E''| \geq (k-1)^2+1$. Order the edges in E''

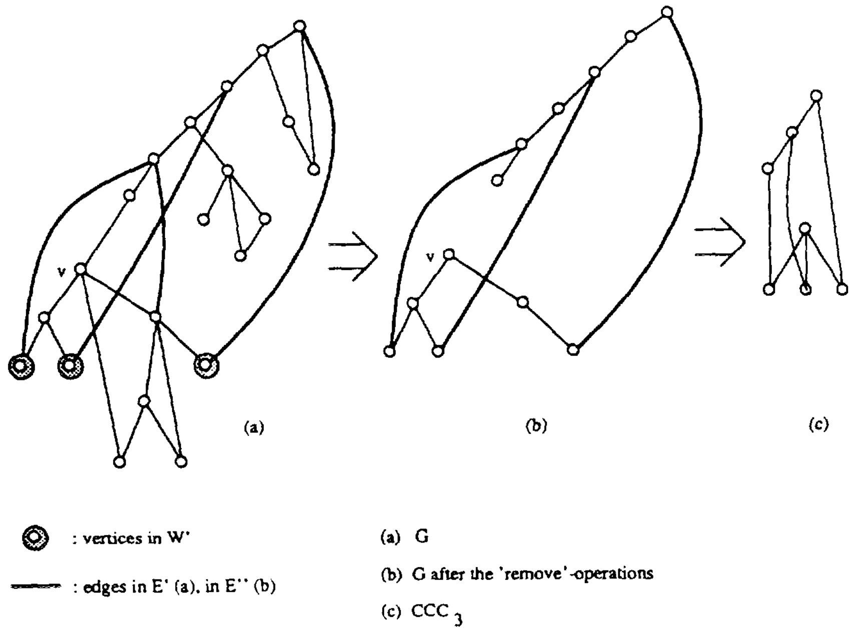
v
v
(a)
(b)
(c)
: vertices in W'
: edges in E' (a), in E'' (b)
(a) G
(b) G after the 'remove'-operations
(c) CCC$_3$

FIG. 2. Construction of the lth circus graph as a minor.

with respect to increasing distance to the root of T of their higher endpoint. Consider the sequence of numbers we obtain by taking the distance to the root of T of the lower endpoints of the edges in E'', in the just-obtained order. This is a sequence of $\geq (k-1)^2 + 1$ different numbers. By the result of Erdös and Szekeres (Theorem 2.5) this sequence has an increasing subsequence of length k, or a decreasing subsequence of length k. Let E''' be a set of k edges, corresponding to such a subsequence.

Now remove all nontree edges not in E'''. Remove all vertices (and adjacent edges) that are not an endpoint of an edge in E''' or a descendant of an endpoint of an edge in E'''. Remove all vertices (and adjacent edges) that are not an endpoint of an edge in E''' or a predecessor of an endpoint of an edge in E'''. Note that all remaining vertices are on one path in (the remainder of) T. We are in a situation, similar to one, illustrated in Fig. 3 (depending on whether the subsequence is increasing or decreasing).

Remove the edge from v to its father. Repeat the following operation, until it is not possible. Take a vertex w that is not adjacent to an edge in E'''. Contract w to one of its neighbors. Now we end up with a graph, isomorphic to the 2×2 grid. □

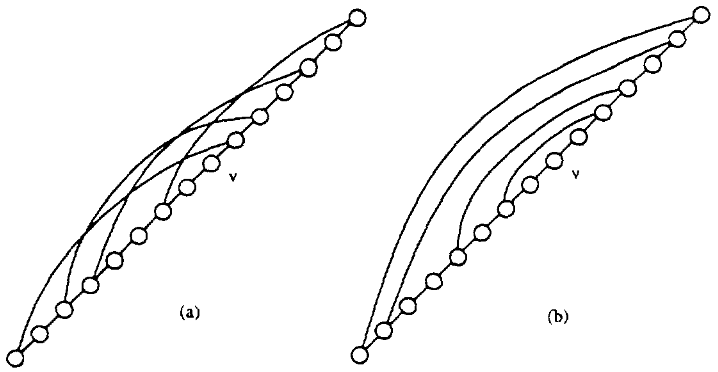

FIG. 3. (a) E''', corresponding to an increasing subsequence. (b) E''', corresponding to a decreasing subsequence.

Let k, l be fixed constants. We give now a procedure that either outputs that a given connected graph $G = (V, E)$ contains a $2 \times k$ grid graph or the lth circus graph as a minor or outputs a T-based tree decomposition of G with treewidth $\leq 2(k-1)^2(l-1)+1$, with T a depth-first search spanning tree of G. Our first version of the procedure uses time that is linear in the number of edges of G. Later we give a small modification that yields a procedure that uses time that is linear in the number of vertices of G. (Further references to "linear time" mean linear in the number of edges, unless specifically stated as linear in the number of vertices.)

1. Traverse G with depth-first search. Assign to each $v \in V$ a new number $\text{dfs}(v) \in \{1, 2, \ldots, |V_G|\}$, in the order they are visited by the traversal. Compute for each $v \in V$: $\text{ldfs}(v) = \max\{\text{dfs}(w) | w = v \text{ or } w \text{ is descendant of } v\}$. Let $T = (V, F)$ be the corresponding dfs-spanning tree of G.

Remark. It is easy to compute $\text{ldfs}(v)$ for all $v \in V$ in linear time. Note that w is a descendant of v, if and only if $\text{dfs}(v) < \text{dfs}(w) \leq \text{ldfs}(v)$.

2. Build a table "back," that contains for each $v \in V$ a pointer to a double linked list that contains, in order of increasing dfs-numbers, the successors of v that have a back edge to v.

Remark. The table can be built in linear time by visiting the vertices $v \in V$ in order of increasing dfs-number. When visiting a vertex v, for every back edge (v, w) that goes from v to a predecessor w, one has to put v at the end of the linked list back(w).

3. Now for each vertex $v \in V$, a set $F(v)$ of nontree edges around v is computed, until all sets $F(v)$ are computed, or a set $F(v)$ is found with $|F(v)| \geq (k-1)^2(l-1)+1$. Each set $F(v)$ forms a matching of edges around v, i.e., no two edges in a set $F(v)$ have an endpoint in common. The sets $F(v)$ are computed in order of increasing dfs-number. As soon as a set $F(v)$ is found, with size at least $(k-1)^2(l-1)+1$, then the computation of the sets $F(v)$ is halted. (In this case, we know by Lemma 3.1 that G contains a $2 \times k$ grid graph or the lth circus graph as a minor.) The sets $F(v)$ are computed as follows:

(a) If dfs(v) = 1, then $F(v) = \emptyset$. (I.e., v is the root of dfs-tree T.)

(b) If dfs(v) $\neq$ 1, then suppose w = father(v). Because dfs(w) < dfs(v), we have already computed the set $F(w)$. Now execute:

```
(1)  $F(v) := \emptyset$
(2)  for each $(x, y) \in F(w): (y$ is successor of $x)$
(3)  do if $y$ is a successor of $v$ or $y = v$
(4)     (i.e., $\mathrm{dfs}(v) \leq \mathrm{dfs}(y) \leq \mathrm{ldfs}(v)$)
     Here, all edges in $F(w)$ that are around $v$ are put into $F(v)$.
(5)     then $F(v) := F(v) \cup \{(x, y)\}$
(6)     else
     Try to find an edge around $v$ with one endpoint $x$, and the other
     endpoint $z$ not endpoint of another edge in $F(v)$.
     Choose $(x, z)$ such that $z$ has maximal depth in $T$.
(7)          if there exists a vertex z with
(8)              (i) $\neg\exists u$: $(u, z) \in F(v) \cup F(w)$
(9)          and (ii) $z$ is a successor of $v$ or $z = v$
(10)         and (iii) $(x, z) \in E - F$
(11)         then choose such a vertex $z$, that does not have a descendant
(12)             $z'$, also fulfilling the properties;
(13)             $F(v) := F(v) \cup \{(x, z)\}$
(14)         endif
(15)    endif
(16) enddo
     Try to find an edge around $v$ with one endpoint $v$, and the other
     endpoint $y$ not endpoint of another edge in $F(v)$.
     Choose $(v, z)$ such that $z$ has maximal depth in $T$.
(17) if there exists a vertex $z$ with
(18)    (i) $\neg\exists u(u, z) \in F(v) \cup F(w)$
(19)    (ii) $z$ is a successor of $v$ or $z = v$
(20)    (iii) $(w, z) \in E - F$
```

```
(21) **then** choose such a vertex $z$, that does not have a descendant $z'$,
(22)     also fulfilling the properties;
(23)     $F(v) := F(v) \cup \{(w, z)\}$
(24) **endif**
```

Remark. After the remaining description of the algorithm, we show that this step can be implemented in $\mathcal{O}(e)$ time. Several properties of the sets $F(v)$ will be proved.

4. If we have found a set $F(v)$, with $|F(v)| \geq (k-1)^2(l-1)+1$, then output "$G$ contains $GR_{2\times k}$ or CC_l as a minor." Otherwise, for all v, with $\text{dfs}(v) \neq 1$: if w is the father of v in T, let $X_v = \{v, w\} \cup \{x \in V \mid (\exists y \in V\colon (x, y) \in F(v) \cup F(w) \vee (y, x) \in F(v) \cup F(w))$ and x is a predecessor of v or x is a successor of $v\}$. For the root r of T (i.e., $\text{dfs}(r) = 1$), let $X_v = \{r\}$.

We claim that $(\{X_v | v \in V\}, T)$ is a T-based tree decomposition of G with treewidth $\leq 2(k-1)^2(l-1)+1$. This will be shown with help of a numbeı of lemmas.

LEMMA 3.2. (i) *If $(x, y) \in F(v)$, then v is on the path from x to y in T, and $(x, y) \in E - F$.*

(ii) *If $(v, x) \in F(v)$, then x is a predecessor of v.*

Proof. This follows directly from the construction of $F(v)$. □

LEMMA 3.3. *For all $v \in V$, $F(v)$ is a matching of the subgraph of G consisting of the nontree edges $(w, x) \in E - F$ with v on the path from w to x in T.*

Proof. This follows from the construction of the sets $F(v)$. Note that we cannot add an edge (w, x) (w the father of v) in line 5 or 13 of the program. (Use Lemma 3.2(ii). □

LEMMA 3.4. *If there exists a $w \in V$ with $|F(v)| \geq (k-1)^2(l-1)+1$, then G contains $GR_{2\times k}$ as a minor or G contains the lth circus graph as a minor.*

Proof. This is a direct corollary of Lemma 3.1 and 3.3. □

LEMMA 3.5. *Step 3 can be implemented, such that the total time needed for this step is bounded by $\mathcal{O}(e)$.*

Proof. First we remark that tests whether a vertex y is a successor of a vertex v can be implemented by "$\text{dfs}(v) < \text{dfs}(y) \leq \text{ldfs}(v)$," and hence cost $\mathcal{O}(1)$ time per test.

We now show how to implement lines 7–14 of step 3, such that these take $\mathcal{O}(e)$ time, in total over all executions of these.

For each $x \in V$ we keep a pointer $p(x)$, which, during the computation of $F(v)$ will point to the first vertex w in the list back(x) with dfs(w) $\geq$ dfs(v). As the lists back(x) are sorted with respect to dfs-numbers and the computations of $F(v)$ are done in order of increasing dfs-number of v, it easily follows that one can update the pointers p in between each two successive computations of a set $F(v)$, such that the total cost of all these updates is bounded by $O(e)$. Note that the edge that must be found will be in the list back(x) at or after position $p(x)$. We must have that dfs(z) $\leq$ ldfs(v); hence we do not have to inspect the part of back(x) that contains the edges (x, y) with dfs(y) $>$ ldfs(v). Implement lines 7–14 as follows:

1. Let pointer $q := p(x)$.

2. If q = **nil**; (we are at the end of list back(x)), then **exit**. (*There are no (further) entries to inspect in list* back(x), *hence no vertex z, fulfilling properties* (i), (ii), (iii) *exists.*)

3. Suppose q points to back edge (x, z), z a successor of x.

4. If dfs(z) $>$ ldfs(v), then **exit**. (*There are no (further) entries in list* back(x) *that can contain a vertex z that has properties* (i), (ii), (iii), *hence such a vertex z does not exist.*)

5. If there exists a vertex $u \in V$ with $(u, z) \in F(v) \cup F(w)$, then let q point to the next item in list back(x); go to step 2. (*z does not fulfill property* (i).)

6. (*Now z is a candidate: it fulfills properties* (i), (ii), (iii). *We now try to find a descendant of z that also fulfills these properties.*) Let q point to the next item in list back(x).

7. If q = nil, then: $F(v) := F(v) \cup \{(x, z)\}$; **exit**. (*There are no further entries to inspect in list* back(x); *hence z has no descendant that also fulfills properties* (i), (ii), (iii).)

8. Suppose q points to back edge (x, z').

9. If dfs(z') $>$ ldfs(z), then $F(v) := F(v) \cup \{(x, z)\}$; **exit**. (*All further entries in list* back(x) *point to edges (x, z''), where* dfs(z'') $>$ *ldfs*(z); hence z'' is not a descendant of z. *So z does not have a descendant fulfilling the properties*; *hence we may add (x, z) to $F(v)$.*)

10. If there exists a vertex $u \in V$ with $(u, z') \in F(v) \cup F(w)$, then let q point to the next item in list back(x); go to step 7. (*z' does not fulfill property* (i).)

11. Remove z from the list back(x). (*See below why we are allowed to do this.*)

12. (*z' is a descendant of z that fulfills properties* (i), (ii), (iii).) $z' := z$. Go to step 6.

We claim that these steps indeed implement lines 7–14 of the original procedure correctly and use in total, $\mathcal{O}(e)$ time. Note that in step 4, if $\mathrm{dfs}(z) > \mathrm{ldfs}(v)$, then z is not a successor of v, $z \neq v$, and neither are any of the vertices that are after z on the list $\mathrm{back}(x)$. Hence we correctly conclude that no vertex z, fulfilling the properties (i), (ii), (iii) exists. In step 6 we know that $\mathrm{dfs}(z) \geq \mathrm{dfs}(v)$ (as q starts at $p(x)$, i.e., on a position with $\mathrm{dfs}(z) \geq \mathrm{dfs}(v)$, and then moves only to vertices with higher dfs-number), and that $\mathrm{dfs}(z) \leq \mathrm{ldfs}(v)$. Hence z fulfills property (ii). It also fulfills property (i), (iii) as can easily be seen. In steps 6–12 we look for a descendant of z that also fulfills properties (i)–(iii). If we know that such a descendant does not exist, then we add (x, z) to $F(v)$ (steps 7, 9). The argument for the correctness of step 9 is similar to above. If we find a descendant z' of z that also fulfills properties (i) to (iii), then there does not exist a vertex v' with $\mathrm{dfs}(v') \geq \mathrm{dfs}(v)$, and $(x, z) \in F(v')$. Suppose $(x, z'') \in F(v)$, z'' a descendant of z. Then, by construction, $(x, z'') \in F(v')$ for all v' on the path from v to z. By Lemma 3.3. $(x, z) \notin F(v')$ for all v' on the path from v to z. Now, by Lemma 3.2(i), $(x, z') \notin F(v')$ for all v' with $\mathrm{dfs}(v') \geq \mathrm{dfs}(v)$. As the edge (x, z) will not belong to any set $F(v')$ that will be computed later, we are indeed allowed to remove z from the list $\mathrm{back}(x)$ (step 11). In step 12 z' becomes the new candidate, and we repeat, looking for a descendant that also fulfills properties (i)–(iii).

The procedure uses time linear in the number of edges. First, note that the test in step 5 or 10 costs constant time, as the size of set $F(w)$, and hence the size of set $F(v)$ is bounded by a constant. These tests are at most $|F(v) \cup F(w)| = \mathcal{O}(1)$ times **true** in one execution of the procedure described above. Hence, the total number of times over all executions of this procedure, that a "loop" is caused by one of these tests is $\mathcal{O}(n)$. The other case where we can loop is if we reach step 11/12. Here we delete an edge from a list $\mathrm{back}(x)$. Hence the total number of this type of loops in $\mathcal{O}(e)$. As $\mathrm{back}(x)$ is a double linked list, step 11 costs $\mathcal{O}(1)$ time. It follows that the total time of this implementation is $\mathcal{O}(e)$.

Lines 17–24 can be implemented in the same way as lines 7–14. □

LEMMA 3.6. *Let $(x, y) \in E - F$ be a nontree edge around v, x a predecessor of y, and let $v \notin \{x, y\}$. Then there exists a vertex $z \in V$, such that $(x, z) \in F(v)$ or $(z, y) \in F(v)$.*

Proof. Use induction on the distance from x to v. If v is a child of x, then consider two cases:

Case 1. There exists a vertex $u \in V$ with $(u, y) \in F(x)$. Then, by construction, $(u, y) \in F(v)$.

Case 2. There does not exist a vertex $u \in V$ with $(u, y) \in F(x)$. Then, as $(x, y) \in E - F$, a vertex z is chosen, and $F(v) := F(v) \cup \{(x, z)\}$ is executed.

Next suppose v is not a child of x. Let w be the father of v. By induction hypothesis, there exists a vertex $z \in V$, with $(x, z) \in F(w)$ or $(z, y) \in F(w)$. We consider two cases:

Case 1. There exists a vertex $z \in V$ with $(z, y) \in F(w)$. Then, by construction, $(z, y) \in F(v)$.

Case 2. There does not exist a vertex $z' \in V$ with $(z', y) \in F(w)$ and there exists a vertex $z \in V$ with $(x, z) \in F(w)$. If z is a successor of v or $z = v$, then, by construction, $(x, z) \in F(v)$. If z is not a successor of v and $z \neq v$, then (x, y) fulfills properties (i)–(iii) (lines 7–9). Hence a vertex z' is chosen and $F(v) := F(v) \cup \{(x, z')\}$ is executed. □

LEMMA 3.7. *For all $(x, y) \in E$: there exists a vertex $v \in V$: $x, y \in X_v$.*

Proof. If (x, y) is a tree edge, x is the father of y, then, by construction, $x, y \in X_y$.

Next suppose (x, y) is not a tree edge, y is a descendant of x. By Lemma 3.6, one of the following three cases must hold:

Case 1. For each vertex $v \neq x, y$ on the path from x to y in T, there exists a vertex $z \in V$ with $(z, y) \in F(v)$. Then let v' be the first vertex after x on this path. Now, by construction, $x \in X_{v'}$, and $y \in X_{v'}$.

Case 2. For each vertex $v \neq x, y$ on the path from x to y in T, there exists a vertex $z \in V$ with $(x, z) \in F(v)$. Then, by construction, $x \in X_y$ and $y \in X_y$.

Case 3. There exists a vertex $v_1 \neq x, y$ on the path from x to y in T, such that there exists a vertex $z \in V$ with $(z, y) \in F(v_1)$, and there exists a vertex $v_2 \neq x, y$ on the path from x to y in T, such that there exists a vertex $z \in V$ with $(x, z) \in F(v_2)$. Then there exists also such vertices v_1, v_2 that are adjacent by a tree edge. If v_1 is the father of v_2, then $x, y \in X_{v_2}$, otherwise, $x, y \in X_{v_1}$. □

LEMMA 3.8. (i) *If x is a predecessor of w, and there exists a vertex $z \in V$ with $(z, w) \in X_x$, then for all y on the path from x to w: $(z, w) \in X_y$.*

(ii) *If w is the father of v in T, then $\{x | x$ is a predecessor of w and there exists a vertex $z \in V$ with $(x, z) \in F(w)\} \supseteq \{x | x$ is a predecessor of w and there exists a vertex $z \in V$ with $(x, z) \in F(v)\}$.*

(iii) *If x is a successor of w and there exists a vertex $z \in V$ and $(w, z) \in X_x$, then for all y on the path from w to x, $y \neq w : (w, z) \in X_y$.*

Proof. This follows from the construction of the sets $F(v)$. □

LEMMA 3.9. *For all* $v \in V$: *the set* $\{w \in V | v \in X_w\}$ *forms a connected subtree of* T.

Proof. This follows from Lemma 3.8 and the construction of the sets X_w. □

Lemmas 3.7 and 3.9 show that $(\{X_v | v \in V\}, T)$ is indeed a tree-decomposition of G. As for all $v \in V$: $|F(v)| \leq (k-1)^2(l-1)$, it directly follows that the treewidth of this tree decomposition is bounded by $4(k-1)^2(l-1)+1$. A better estimate is possible by a more detailed analysis.

LEMMA 3.10. *If for all* $v \in V$: $|F(v)| \leq c$, *then for all* $v \in V$: $|X_v| \leq 2c+2$.

Proof. Consider v with $\mathrm{dfs}(v) \neq 1$, i.e., v is not the root of T. Consider $x \in X_v$, x a predecessor of v, and x is not the father of v in T. Let w be the father of v. Then there exists a vertex $z \in V$ with $(x,z) \in F(w)$ or $(x,z) \in F(v)$. By Lemma 3.8(ii), we have that there exists a vertex $z \in V$ with $(x,z) \in F(w)$. Hence $|\{x \in X_v | x \text{ is a predecessor of } v\}| \leq |F(w)| + 1 \leq c+1$.

Next consider $x \in X_v$, x a successor of v. Let w be the father of v. Now there exists a vertex $z \in V$ with $(z,x) \in F(w)$ or $(z,x) \in F(v)$. By construction, $(z,x) \in F(w) \Rightarrow (z,x) \in F(v)$. Hence $|\{x \in X_v | x \text{ is a successor of } v\}| \leq |F(v)| \leq c$.

By construction we have $x \in F(v) \Rightarrow x$ is a predecessor of v or $x = v$ or x is a successor of v. Hence $|X_v| \leq 2c+2$. □

THEOREM 3.11. *For all* $k, l \geq 1$, *for all graphs* $G = (V,E)$, G *contains the* $2 \times k$ *grid as a minor or* G *contains the* l*th circus graph as a minor or treewidth* $(G) \leq 2(k-1)^2(l-1)+1$.

Proof. This follows from Lemmas 3.4, 3.7, 3.9, and 3.10. □

From Lemma 2.4. and Theorem 3.11 it follows that we can use the following variant of the algorithm. This variant uses $\mathcal{O}(n)$ time.

Test whether $|E| > (2(k-1)^2(l-1)+1)|V| + \frac{1}{2}(2(k-1)^2(l-1)+2)(2(k-1)^2(l-1)-1)$.

If this is the case, then treewidth $(G) > 2(k-1)^2(l-1)+1$; hence, by Lemma 3.11, G contains $GR_{2\times k}$ or the lth circus graph as a minor.

If this is not the case, run the original algorithm. (Now $|E| = \mathcal{O}(|V|)$.)

THEOREM 3.12. *Let* k, l *be constants. There exists an algorithm, that given a graph* $G = (V,E)$, *either outputs that "*G *contains the* $2 \times k$ *grid or the* lth *circus graph" as a minor; or outputs a tree decomposition of* G *with treewidth* $\leq 2(k-1)^2(l-1)+1$, *and that uses* $\mathcal{O}(|V|)$ *time.*

4. Further Results

In this section we give a number of results that follow from Theorems 3.11 and 3.12 or that show to what extent the techniques can be used.

Corollary 4.1. *Let A be a minor-closed class of graphs and suppose that $GR_{2\times k} \notin A$ and $CC_l \notin A$. Then there exists an algorithm, that given a graph G, decides in $\mathcal{O}(n)$ time whether $G \in A$ or not.*

Proof. First run the algorithm, indicated by Theorem 3.12. If this algorithm yields a tree decomposition of G with treewidth $\leq 2(k-1)^2(l-1)+1 = \mathcal{O}(1)$, then test whether G contains an element of the obstruction set of A as a minor in linear time (cf. Theorem 2.1). □

Corollary 4.2. *Let H be a minor of the $2 \times k$ grid graph, and the lth circus graph. Then, for all graphs G that do not contain H as a minor, treewidth $(G) \leq 2(k-1)^2(l-1)+1$.*

Proof. This is a direct corollary of Theorem 3.11. □

Corollary 4.3. *Let H be a minor of the $2 \times k$ grid graph, and the lth circus graph. Then there exists an algorithm, that given a graph G, decides in $\mathcal{O}(n)$ time whether G contains H as a minor.*

Proof. The class of graphs that do not contain H as a minor is closed under taking of minors, and does not contain the $2 \times k$ grid graph and the lth circus graph. So we can apply Corollary 4.1. □

The class of graphs that are a minor of a $2 \times k$ grid graph of a circus graph includes the following graphs:

- all paths (Fig. 4a)
- all cycles (Fig. 4b)
- all graphs that consists of two cycles that have exactly one edge in common (Fig. 4c)
- all caterpillars with hairs of length 1 (Fig. 4d) (I.e., all trees that contain a path, such that every vertex lies on the path or is adjacent to a vertex on the path.)
- all graphs, that consist of a number of paths and cycles, that have exactly one vertex in common; i.e., all connected graphs where at most one vertex has degree ≤ 3 (Fig. 4e).
- all subgraphs of these.

There are several other examples. For each of these graphs one can test in linear time whether it is a minor of a given graph. If a minor-closed class of graphs avoids one of these graphs, then membership in that class

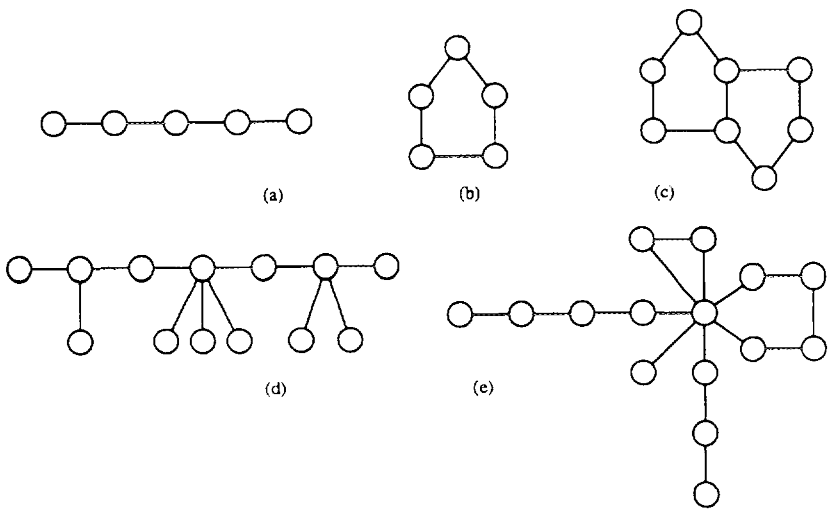

FIG. 4. Graphs that are a minor of a $2 \times k$ grid and of a circus graph.

can be tested in linear time. These results for paths and cycles were already obtained by Fellows and Langston [17] and are discussed in Section 5. We also have the following interesting corollary.

MAXIMUM LEAF SPANNING TREE is the problem to determine, given a graph G and an integer k, whether G contains a spanning tree in which k or more vertices have degree 1. It is NP-complete for variable k [18]. Fellows and Langston [15] have shown that this problem, for fixed k, is solvable in $\mathcal{O}(n^2)$ time, as the set of connected graphs that are a no-instance are closed under minor-taking.

LEMMA 4.4. *G contains a spanning tree with $\geq k$ vertices with degree* 1, *if and only if G is connected and it contains the graph $K_{1,k}$ as a minor.*

Proof. (Note that $K_{1,k}$ is the "k-star" graph, see, e.g., Fig. 5.) If G contains a spanning tree with $\geq k$ leaves, then clearly G is connected. It

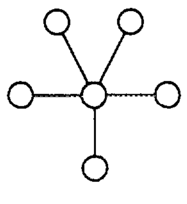

FIG. 5. $K_{1,5}$.

contains $K_{1,k}$ as a minor: remove all nontree edges from G, then contract over every edge between two nonleaf vertices. Now suppose that G is connected and contains $K_{1,k}$ as a minor. One may assume that one can obtain $K_{1,k}$ from G by first choosing a spanning tree of G and then applying a number of contractions. This spanning tree must have $\geq k$ leaves. □

THEOREM 4.5. *For fixed k, the MAXIMUM LEAF SPANNING TREE problem can be solved in $\mathcal{O}(e)$ time, and in $\mathcal{O}(n)$ time, when restricted to connected graphs.*

Proof. Use Lemma 4.4, Corollary 4.3, and note that $K_{1,k}$ is a minor of $GR_{2\times k}$ and of CC_k. □

A more efficient algorithm can be obtained in this case with a construction, that is a small variant of the construction of Section 3. Let $X_v = \{v\} \cup \{w \mid w$ is a predecessor of v, and there exists a vertex $x \in V$ with (w,x) a nontree edge around $v\}$. As soon as one finds a $v \in V$ with $|X_v| \geq k+1$, then stop; in this case G contains $K_{1,k}$ as a minor. The remaining algorithm is similar to the general case. We omit the details. In this way one uses tree decompositions with treewidth at most k, instead of $(k-1)^3+1$.

By making small modifications to the algorithm, one can turn it into one that also constructs the spanning tree, if it exists, and that also uses $\mathcal{O}(n)$ time. A similar result was obtained independently by Fellows [11].

Another, but perhaps less interesting corollary of our results is the fact that there exists an $\mathcal{O}(n)$ algorithm to decide whether a given graph has search number ≤ 2. (The class of graphs with search number ≤ 2 is closed under minor-taking and avoids $GR_{2\times 3}$ and CC_3.) (For the definition of search number, see, e.g., [23].) This result has already been obtained by Megiddo *et al.* [23] in a different, more direct way.

The following results show that our results are optimal in the sense that we cannot deal with other graphs/classes of graphs with the same method. Note that these results do not state anything about the possibility of obtaining linear time minor test algorithms with depth-first search, in general, but only give the limitations of the specific approach used in this paper, namely, to look for T-based tree-decompositions, where T is an arbitrary depth-first search spanning tree.

LEMMA 4.6. *Suppose there exists a constant C_H, such that for all connected graphs G that do not contain H as a minor, for all depth-first search spanning trees T of G there exists a T-based tree decomposition of G*

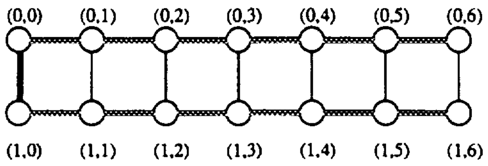

FIG. 6. $GR_{2\times 6}$ and T.

with treewidth $\leq C_H$. *Then*

(i) *There exists a* k, *such that* H *is a minor of the* $2 \times k$ *grid graph,*

(ii) *There exists an* l, *such that* H *is a minor of the* l*th circus graph.*

Proof. (i) H is a minor of the $2 \times (C_H + 3)$ grid graph. Suppose not. Take $G = GR_{2\times(C_H+3)}$. Consider the depth-first search spanning tree T, obtained by starting the dfs-traversal in $(0, C_H + 2)$, then visiting $(0, i)$ for $i = C_H + 1, C_H, \ldots, 2, 1, 0$, then visiting $(1, 0)$, $(1, i)$ for $i = 1, 2, \ldots, C_H + 2$ (see Fig. 6).

Consider an arbitrary T-based tree decomposition $(\{X_v | v \in V_G\}, T)$ of G. For each edge $((0, i), (1, i))$ with $1 \leq i \leq C_H + 2$, note that $(0, 0)$ is on the path from $(0, i)$ to $(1, i)$ in T. Hence, by Lemma 2.3, $(0, i) \in X_{(0,0)}$ or $(1, i) \in X_{(0,0)}$. So $|X_{(0,0)}| \geq C_H + 2$. It follows that every T-based tree decomposition of G has treewidth $\geq C_H + 1$, a contradiction.

(ii) With a similar proof one can show that H is a minor of the $(C_H + 3)$th circus graph. Suppose not. Take $G = CC_{C_H+3}$, and use the spanning tree T, with edges (x, y_i) $(i = 1, \ldots, C_H + 3)$, (y_1, z_1) and (z_i, z_{i+1}) $(i = 1, \ldots, C_H + 2)$ (see Fig. 7). This is a depth-first search spanning tree with root z_{C_H+3}. From Lemma 2.3, it follows that $|X_x| \geq C_H + 2$. □

Combining Lemma 4.6 and Theorem 3.11 gives

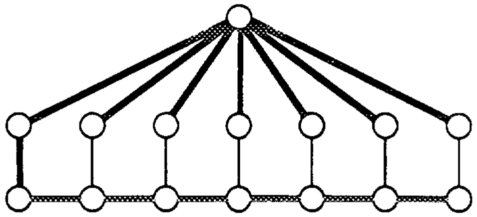

FIG. 7. CC_7 and T.

COROLLARY 4.7. *The following two statements are equivalent:*

(i) *There exists k, l such that H is a minor of $GR_{2\times k}$ and of CC_l.*

(ii) *There exists a constant C_H, such that for all connected graph G, that do not contain H as a minor, for all depth-first search spanning trees T of G, there exists a T-based tree decomposition of G with treewidth $\leq C_H$.*

In the same way one can prove:

LEMMA 4.8. *Let A be a minor closed class of graphs. Suppose that A contains all graphs $GR_{2\times k}$ ($k \in \mathbf{Z}^+$), or A contains all graphs CC_l ($l \in \mathbf{Z}^+$). Then for every $m \in \mathbf{Z}^+$, there exists a connected graph $G \in A$, such that G has a depth-first search spanning tree T, such that every T-based tree decomposition of G has treewidth $> m$.*

5. FINAL REMARKS

5.1. In [17] Fellows and Langston gave an algorithm that decides in linear time whether a graph $G = (V, E)$ contains a cycle (or path) with length $\geq k$ (k fixed). In [17] they show that these cycles (paths) can be constructed by self-reduction algorithms, that use these decision algorithms as an oracle, in $\mathcal{O}(n^2 \log n)$ ($\mathcal{O}(n \log n)$) time. However, with small modifications one can turn the decision algorithm in a construction algorithm that also uses $\mathcal{O}(n)$ time. This result improves on a result of Monien [24], who obtained $\mathcal{O}(ne)$ algorithms for the problems. However, this latter result is also valid for directed graphs, whereas the "depth first search approach" seems unsuitable for directed graphs. (The main problem is that one can have "cross-edges" with depth first search in a directed graph). Monien [24] also obtained $\mathcal{O}(ne)$ algorithms to find cycles of length exactly k. The following algorithm is a small modification of an algorithm of Fellows and Langston [17]. It finds a cycle with length $\geq k$ in a given graph G, if it exists.

1. Start a depth first search traversal of G. When we reach a vertex, that was not visited before, compute its distance to the root (by adding one to this number of its father). If we traverse a back edge, then compare the distances to the root of its endpoints. If this difference is $k - 1$ or larger, then stop the traversal, and output the cycle, consisting of this edge and the path in the dfs-tree between its endpoint. This cycle has length $\geq k$.

2. If the dfs-traversal is completed, (we did not already output a cycle), and dfs-tree T is constructed, then let for all $v \in V$: X_v is the set, containing v and its $k - 2$ direct predecessors of T (or all its predecessors). Then $(\{X_v | v \in V\}, T)$ is a T-based tree decomposition of G with treewidth $\leq k - 2$. Now apply Theorem 2.2.

A similar algorithm can be used if we want to find a path in G with length $\geq k$.

THEOREM 5.1. (i) *There exists an algorithm, that given a graph $G = (V, E)$ and an integer k, either finds a simple cycle (path) in G with length $\geq k$, or determines that such a cycle (path) does not exist in $\mathcal{O}(2^k k!n)$ time.*

(ii) *The* LONGEST CYCLE (LONGEST PATH) *problem can be solved in $\mathcal{O}(2^{\mu}(\mu + 1)!n)$ time, where μ is the length of the longest cycle (path).*

We believe that these algorithms are practical for small values of k and may, in some cases, be a good alternative for branch-and-bound algorithms or the algorithms of Monien [24].

We now comment on how to find the longest path between two specified vertices s, t. We use the following lemma.

LEMMA 5.2. *Let $G = (V, E)$ be a biconnected graph, containing a cycle with length $2k - 1$. Let $s, t \in V$. Then there is a simple path from s to t with length $\geq k$.*

Proof. Let e be an arbitrary edge on the cycle. By biconnectivity, there exists a simple path p from s to t that uses e. Let v be the first vertex on p, that lies on the cycle, and let w be the last such vertex. Necessarily $v \neq t$. Now use the following path from s to t; use p from s to v, then go from v to w using the largest of the two routes from v to w over the cycle, and then use p again to go from w to t. This path is simple and has length $\geq k$. □

Observe that it is sufficient to have an algorithm for biconnected graphs: for each biconnected component of G, either no path between s and t visits this component, or every path between s and t enters and leaves the component at the same two vertices. Components of the former type can be ignored; components of the latter type can be considered separately.

Suppose now that G is biconnected. Make a depth-first search traversal of G. If no back edge goes between vertices with distance $\geq 2k - 1$ in T, then one can build a T-based tree decomposition of G with treewidth $\leq 2k - 1$ similarly as before, and then solve the problem (as in Theorem 2.2) in $\mathcal{O}((2k)!2^{2k}n)$ time. If there exists a back edge between vertices with distance $\geq 2k - 2$ in T, then G contains a cycle with length $\geq 2k - 1$, and hence a path from s to t with length $\geq k$. (One can make the proof of Lemma 5.2 constructive, such that this cycle is found in $\mathcal{O}(e)$ time.)

THEOREM 5.3. (i) *There exists an algorithm that, given a graph $G = (V, E)$, two vertices $s, t \in V$, an integer k, either finds a simple path from s*

to t with length $\geq k$, or determines that such a path does not exist in $\mathcal{O}(2^{2k}(2k)!n + e)$ time.

(ii) *Given a graph $G = (V, E)$ and two vertices $s, t \in V$, one can determine the longest path between s and t in $\mathcal{O}(2^{2\mu}(2\mu + 2)!n + e)$ time, where μ is the length of the longest path between s and t.*

5.2. Lemma 4.6, Corollary 4.7, and Lemma 4.8 show that we cannot extend our results to graphs H that are not a minor of a $2 \times k$ grid graph or not of a circus graph, and to classes A that include all $2 \times k$ grid graphs or all circus graphs, when we insist on using T-based tree decompositions with T an arbitrary depth-first search spanning tree. However, it is conceivable that such extensions are possible when we use spanning trees which are made by some modification of depth-first search, or by depth-first search with some extra requirements. For example, testing whether G contains K_5 or $K_{3,3}$ as a minor (e.g., G is a planar) can also be done in $\mathcal{O}(n)$ time with an algorithm that is based on depth-first search [19]. So, there may be some important connections between graph minors and depth-first search that are not yet discovered.

5.3. The class of graphs that are a minor of a $2 \times k$ grid graph (k variable) and of a circus graph is also closed under minor-taking and hence can be characterized by its obstruction set. However, this obstruction set contains at least 10 graphs and seems not to give any interesting insights in this class.

Acknowledgments

Thanks are due to Mark de Berg and Marc van Kreveld for providing examples of graphs that are minor of a $2 \times k$ grid and of a circus graph, and to Jan van Leeuwen for some useful comments.

References

1. S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese, "An algebraic theory of graph reduction," *in* "Proceedings, 4th Workshop on Graph Grammars and Their Applications to Computer Science," pp. 70–83, Lect. Notes in Comput. Sci., Vol. 532, Springer–Verlag, New York/Berlin, 1991.
2. S. Arnborg, J. Lagergren, and D. Seese, Easy problems for tree-decomposable graphs, *J. Algorithms* **12** (1991), 308–340.
3. S. Arnborg and A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-trees, *Discrete Appl. Math.* **23** (1989), 11–24.
4. H. L. Bodlaender, NC-algorithms for graphs with small treewidth, *in* "Proceedings, Workshop on Graph-Theoretic Concepts in Computer Science WG'88" (J. van Leeuwen, Ed.) pp. 1–10, Lect. Notes in Comput. Sci., Vol. 344, Springer–Verlag, New York/Berlin, 1988.

5. H. L. BODLAENDER, Improved self-reduction algorithms for graphs with bounded treewidth, *in* "Proceedings, 15th Int. Workshop on Graph-theoretic Concepts in Computer Science WG'89," pp. 232–244, Lect. Notes in Comput. Sci., Vol. 411, Springer-Verlag, New York/Berlin, 1990, *Ann. Discrete Math.*, to appear.
6. R. B. BORIE, R. G. PARKER, AND C. A. TOVEY, Automatic generation of linear algorithms from predicate calculus descriptions of problems on recursive constructed graph families, *Algorithmica* **7** (1992), 555–582.
7. D. J. BROWN, M. R. FELLOWS, AND M. A. LANGSTON, Nonconstructive polynomial-time decidability and self-reducibility, *Internat. J. Comput. Math.* **31** (1989), 1–9.
8. B. COURCELLE, "The Monadic Second-Order Logic of Graphs III: Treewidth, Forbidden Minors and Complexity Issues," Report 8852, University of Bordeaux 1, 1988.
9. B. COURCELLE, The monadic second-order logic of graphs I: Recognizable sets of finite graphs, *Inform. and Comput.* **85** (1990), 12–75.
10. P. ERDÖS AND G. SZEKERES, A combinatorial problem in geometry, *Compositio Math.* **2** (1935), 464–470.
11. M. R. FELLOWS, personal communication, 1989.
12. M. R. FELLOWS AND M. A. LANGSTON, Nonconstructive advances in polynomial-time complexity, *Inform. Process. Lett.* **26** (1987), 157–162.
13. M. R. FELLOWS AND M. A. LANGSTON, "Fast Search Algorithms for Graph Layout Permutation Problems," Technical Report CS-88-189, Dept. of Computer Science, Washington State University, 1988.
14. M. R. FELLOWS AND M. A. LANGSTON, Nonconstructive tools for proving polynomial-time decidability, *J. Assoc. Comput. Mach.* **35** (1988), 727–739.
15. M. R. FELLOWS AND M. A. LANGSTON, "On Well-Partial-Order Theory and Its Application to Combinatorial Problems of VLSI Design," Technical Report CS-88-188, Dept. of Computer Science, Washington State University, 1988.
16. M. R. FELLOWS AND M. A. LANGSTON, An analogue of the Myhill–Nerode theorem and its use in computing finite-basis characterizations, *in* "Proceedings, 30th Annual Symposium on Foundations of Computer Science, 1989," pp. 520–525.
17. M. R. FELLOWS AND M. A. LANGSTON, On search, decision and the efficiency of polynomial-time algorithms, *in* "Proceedings, 21st Annual Symposium on Theory of Computing, 1989," pp. 501–512.
18. M. R. GAREY AND D. S. JOHNSON, "Computers and Intractability, A Guide to the Theory of NP-Completeness," Freeman, New York, 1979.
19. J. E. HOPCROFT AND R. E. TARJAN, Efficient planarity testing, *J. Assoc. Comput. Mach.* (1974), 549–568.
20. D.S. JOHNSON, The NP-completeness column: An ongoing guide, *J. Algorithms* **8** (1987), 285–303.
21. J. LAGERGREN, Efficient parallel algorithms for tree-decomposition and related problems, *in* "Proceedings, 31st Annual Symposium on Foundations of Computer Science, 1990," pp. 173–182.
22. C. LAUTEMANN, Efficient algorithms on context-free graph languages, *in* "Proceedings, 15th International Colloquium on Automata, Languages and Programming, 1988," pp. 362–378, Lect. Notes in Comput. Sci., Vol. 317, Springer–Verlag, New York/Berlin, 1988.
23. N. MEGIDDO, S. L. HAKIMI, M. R. GAREY, D. S. JOHNSON, AND C. H. PAPADIMITRIOU, The complexity of searching a graph, *J. Assoc. Comput. Mach.* **35** (1988), 18–44.
24. B. MONIEN, How to find long paths efficiently, *Ann. Discrete Math.* **25** (1985), 239–254.
25. N. ROBERTSON AND P. D. SEYMOUR, Graph minors. XV. Wagner's conjecture, to appear.
26. N. ROBERTSON AND P. D. SEYMOUR, Graph minors. II. Algorithmic aspects of tree-width, *J. Algorithms* **7** (1986), 309–322.

27. N. Robertson and P. D. Seymour, Graph minors. XIII. The disjoint paths problem, manuscript, 1986.
28. P. Scheffler, "Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme," Ph.D. thesis, Akademie der Wissenschaften der DDR, Berlin, 1989.
29. J. van Leeuwen, Graph algorithms, *in* "Handbook of Theoretical Computer Science. A. Algorithms and Complexity Theory," pp. 527–631, North Holland, Amsterdam, 1990.
30. T. V. Wimer, "Linear Algorithms on k-terminal Graphs," Ph.D. thesis, Dept. of Computer Science, Clemson University, 1987.