ON LINEAR TIME MINOR TESTS AND DEPTH FIRST SEARCH

Hans L. Bodlaender

RUU-CS-89-1
January 1989

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

ON LINEAR TIME MINOR TESTS AND DEPTH FIRST SEARCH

Hans L. Bodlaender

Technical Report RUU-CS-89-1
January 1989

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

ISSN:0924-3275

ON LINEAR TIME MINOR TESTS AND DEPTH FIRST SEARCH

Hans L. Bodlaender
Department of Computer Science, University of Utrecht
P.O.Box 80.089, 3508 TB Utrecht, the Netherlands

Abstract

Recent results on graph minors make it desirable to have efficient algorithms, that for a fixed set of graphs $\{H_1, \ldots, H_c\}$, 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 $H \in \{H_1, \ldots, H_c\}$ 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.

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 [18, 19] 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 [19]. As one can test for fixed graphs H in $\mathcal{O}(n^3)$ time whether a given graph G contains H as a minor [18], this shows the existence of a method to test in $\mathcal{O}(n^3)$ membership in minor closed classes of graphs. If A does not contain all planar graphs, then there exist $\mathcal{O}(n^2)$ algorithms [18]. Recently, Fellows and

Langston showed that if A avoids at least one cycle (or minor of a cycle) then there exist $\mathcal{O}(n)$ algorithms [9].

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 the same technique.

Note that the graph minor theorem of Robertson and Seymour is non-constructive. 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 $\mathcal{O}(n), \mathcal{O}(n^2)$ 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 non-constructive elements of the graph minor theorem by using self-reductions [9].

This paper is further 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 [17].

Definition.
Let $G = (V, E)$ be a graph. A tree-decomposition of G is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ 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 $\{i \mid v \in X_i\}$ forms a connected subtree of T.

The treewidth of a tree-decomposition $(\{X_i \mid i \in I\}, T)$ 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. [1].)

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, 3, 4, 5, 13, 20, 22]). 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 [6] or [18].

Theorem 2.2
Let k be a constant. 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.
The algorithm is similar to the dynamic programming algorithms in [3], [4] or [22]. As is usual with dynamic programming algorithms, one can turn the decision algorithm into a construction algorithm with some extra bookkeeping. We omit the details here. □

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

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

Lemma 2.3
Let $(\{X_v \mid v \in V\}, T = (V, F))$ 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 $(x, y) \in V, x \notin X_v, y \notin X_y$. Then as $x \in X_x$, $y \in X_y$, $\{w \mid x \in X_w\}$ and $\{w \mid y \in X_y\}$ 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 $|E| \leq k \cdot |V| + \frac{1}{2}(k+1)(k-2)$.

Proof.
This follows directly from the fact, that every graph with treewidth $\leq k$ is a subgraph of a $k-$tree see e.g. [1, 21]. A $k-$tree with n vertices has exactly $kn + \frac{1}{2}(k+1)(k-2)$ 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 $V_{n \times m} = \{(i,j) \mid 0 \leq i \leq n-1,\ 0 \leq j \leq m-1\}$ and $E_{n \times m} = \{(i_1,j_1),(i_2,j_2)) \mid (i_1,j_1),(i_2,j_2) \in V_{n \times m}$, and $((|i_1 - i_2| = 1 \wedge j_1 = j_2)$ or $(i_1 = j_2 \wedge |j_1 - j_2| = 1))\}$.

2. The l'th circus graph is the graph $CC_l = (\{x\} \cup \{y_i \mid 1 \leq i \leq l\} \cup \{z_i \mid 1 \leq i \leq l\}, \{(x,y_i) \mid 1 \leq i \leq l\} \cup \{(y_1,z_i) \mid 1 \leq i \leq l\} \cup \{(z_i,z_{i+1}) \mid 1 \leq i \leq l-1\})$.

The 2×5 grid graph and the 5'th circus graph are shown, as examples, in fig. 1.

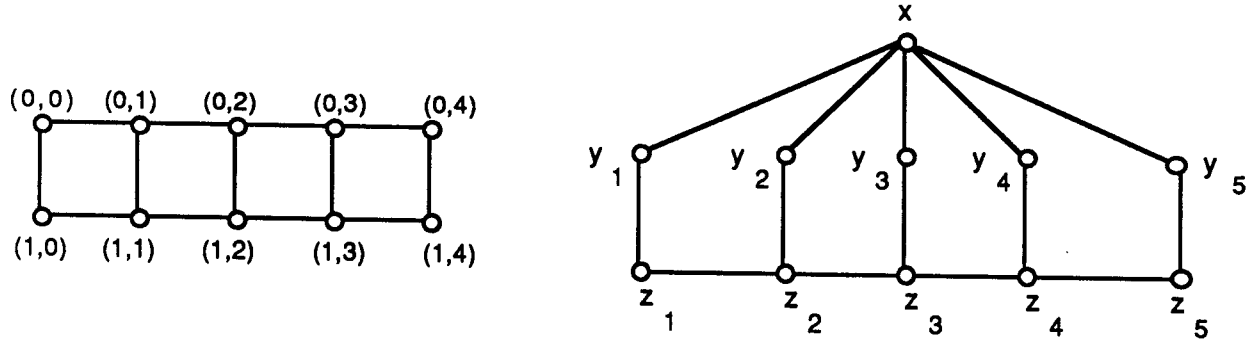

Figure 1: $GR_{2 \times 5}$ and CC_5

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. (See e.g. [15], chapter IV. 4, IV. 5.)

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

Theorem 2.5 (Erdös, Szekeres, 1935 [7])
Let $a_1, \ldots, a_c$ be a sequence of c different numbers, i.e. $1 \leq i < j \leq c \Rightarrow a_i \neq a_j$. If $c \geq (k-1)^2 + 1$, then the sequence $a_1, \ldots, a_c$ has an increasing subsequence of length $\geq k$, or it has a decreasing subsequence of length $\geq k$.

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 the non-tree edges $(w,x) \in E - F$, with w is a predecessor of v, $x = v$ or x is a successor of v, and v is on the path from w to x in T. 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 l'th circus graph as a minor.

Proof.
Let $G = (V, E)$, $T = (V, F)$, v be given. Suppose the subgraph of G, consisting of the non-tree edges $(w, x) \in E - F$, with w a predecessor of v, $x = v$ or x a successor of v, and v on the path from w to x in T 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 do not have a 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 pidgeon-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 l'th 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 non-tree edges from G that are not in E'. Remove all non-tree 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 descendant 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 l'th 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'' 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

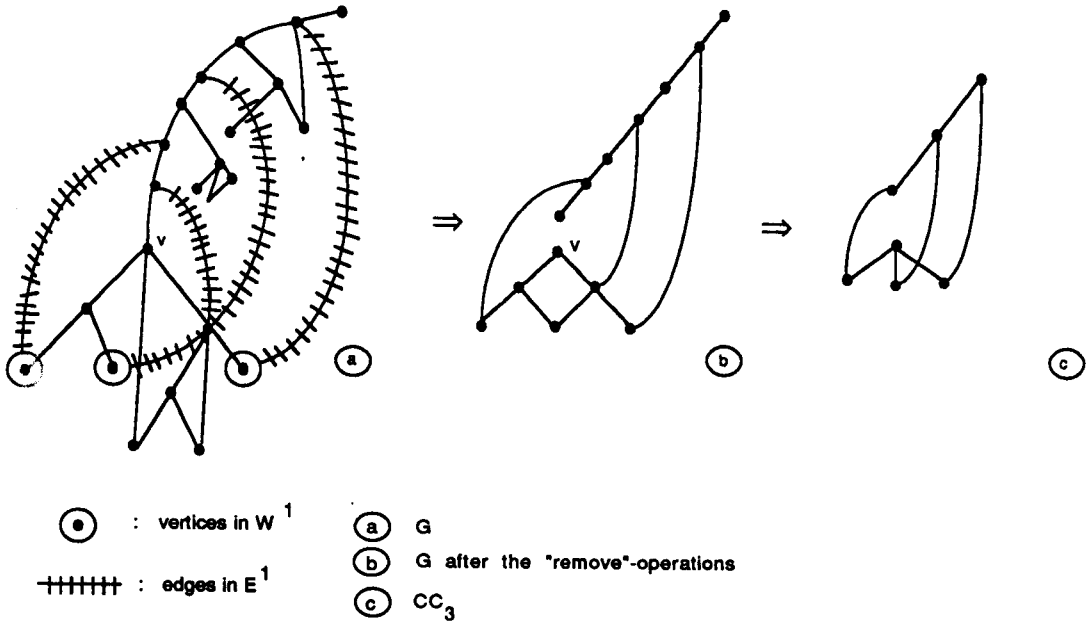

Figure 2: Construction of the l'th circus graph as a minor

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 non-tree-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 $2 \times k$ grid. □

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 l'th 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, linear in the number of vertices of G.

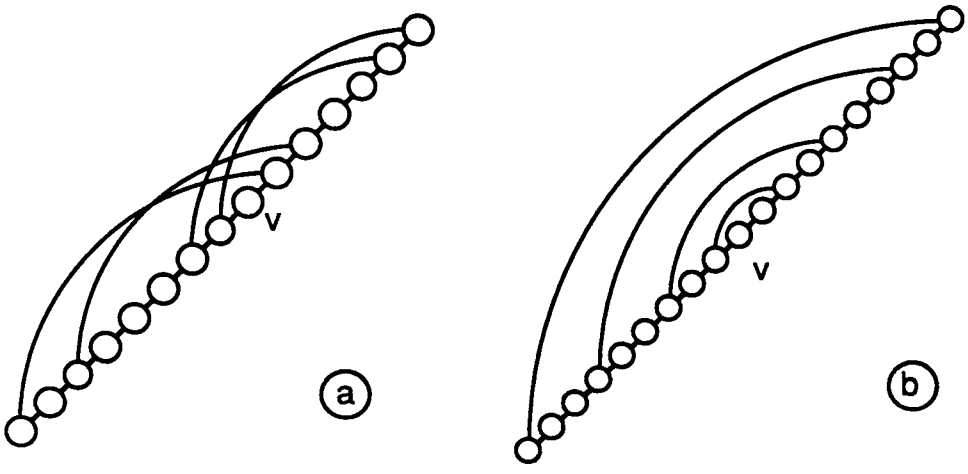

Figure 3: (a) E''', corresponding to an increasing subsequence. (b) E''', corresponding to a decreasing subsequence.

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

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

2. Build a table 'back', that contains for each $v \in V$ a pointer to a linked list that contains, in order to 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. Compute for each $v \in V$ a set $F(v) \subseteq E$ as follows, in order of *dfs*-number of v. However, if we find a set $F(v)$, with $|F(v)| \geq (k-1)^2(l-1)+1$, then stop.
   (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. $dfs(v) \leq dfs(y) \leq ldfs(v)$)
(5) **then** $F(v) := F(v) \cup \{(x, y)\}$
(6) **else if** there exists a vertex z with
(7) (i) $\neg\exists u : (u, z) \in F(w)$
(8) and (ii) z is a successor of v or $z = v$
(9) and (iii) $(x, z) \in E - F$
(10) **then** choose such a vertex z, that does not have a descendant z',
(11) also fulfilling the properties
(12) $F(v) := F(v) \cup \{(x, y)\}$
(13) **endif**
(14) **endif**
(15) **enddo**
(16) **if** there exists a vertex z with
(17) (i)$\neg\exists u(u, z) \in F(w)$
(18) (ii) z is a successor of v or $z = v$
(19) (iii)$(w, z) \in E - F$
(20) **then** choose such a vertex z, that does not have a descendant z',
(21) also fulfilling the properties.
(22) $F(v) := F(v) \cup \{(w, z)\}$
(23) **endif**

Remark. After the remaining description of the algorithm, we show that his 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 let for all v, with $dfs(v) \neq 1$: if w is the father of v in T, then $X_v = \{v, w\} \cup \{x \mid \exists y : (x, y) \in F(v) \cup F(w) \vee (y, x) \in F(v) \cup F(w)$ and x is a predecessor of v or x is a successor of v $\}$. For the root r of t (i.e. $dfs(r) = 1$), let $X_v = \{r\}$.

We claim that $(\{X_v \mid v \in V\}, T)$ 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 number 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 non-tree 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 l'th 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 "$dfs(v) < dfs(y) \leq ldfs(v)$", and hence cost $\mathcal{O}(1)$ time per test.

We now show how to implement lines 6 -14, 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 points to the first vertex w in list back(x) with $dfs(w) \geq dfs(v)$, when we are currently computing $F(v)$. The total cost of updating these pointers is $\mathcal{O}(e)$: we are computing the sets $F(v)$ in order to increasing $dfs(v)$. When we have computed $F(v')$, then we have to adjust the pointers $p(v'')$, for every back edge (v', v''), v'' a predecessor of v. Hence, this costs $\mathcal{O}(e)$ time in total. Implement lines 6-14 as follows:

1. Let pointer $q := p(x)$.
2. If $q =$ nil; (we are at the end of list back(x)), then exit. (No vertex z, fulfilling properties (i), (ii), (iii) exists).
3. Suppose q points to vertex z (i.e. back edge (x, z), z a successor of x).
4. If $dfs(z) > ldfs(v)$, then exit. (No vertex z, fulfilling properties (i), (ii), (iii) exists).
5. If $\exists u : (u, z) \in F(w)$, then let q point to the next item in list back(x); go to step 2.
6. Now z is a candidate: it fulfills properties (i), (ii), (iii). Let q point to the next item in list back(x).
7. If $q =$ nil, then: $F(v) := F(v) \cup \{(x, z)\}$; exit

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

9. If $dfs(z') > ldfs(z)$, then $F(v) := F(v) \cup \{(x, z)\}$; exit.

10. If $\exists u : (u, z') \in F(w)$, then let q point to the next item in list back(x); go to step 7.

11. Remove z from the list back(x).

12. $z' := z$. Go to step 6.

We claim that these steps indeed implement lines (6) – (13) of the original procedure correctly, and use in total $\mathcal{O}(e)$ time. Note that in step 4, if $dfs(z) > ldfs(v)$, 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 back(x). Hence we correctly conclude that no vertex z, fulfilling the properties (i), (ii), (iii) exists. In step 6 we know that $dfs(z) \geq dfs(v)$ (as q starts at $p(x)$, i.e. on a position with $dfs(z) \geq dfs(v)$, and then moves only to vertices with higher *dfs*-number, and that $dfs(z) \leq ldfs(v)$. 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 $F(v)$. (Step 7,9). The argument for the correctness of step 9 is similar as 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 $dfs(v') \geq dfs(v)$, 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 $dfs(v') \geq dfs(v)$. It follows that we are indeed allowed to remove z from the list 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. The tests $\exists u : (u, z) \in F(w), \exists u(u, z') \in F(w)$ are at most $|F(w)| = \mathcal{O}(1)$ times true. Hence, the total number of times 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. Hence we delete an edge from a list back(x). Hence the total number of this type of loops in $\mathcal{O}(e)$. (We need an additional pointer to implement step 11 in $\mathcal{O}(1)$ time. We omit this easy detail.) It follows that the total time of this implementation is $\mathcal{O}(e)$.

Lines 16 – 23 can be implemented in the same way as lines 6 – 13. □

Lemma 3.6
Let $(x, y) \in E - F$, x a predecessor of y, and let $v \notin \{x, y\}$ be on the path from x to y. Then: $\exists z : (x, z) \in F(v) \vee (z, y) \in F(v)$.

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

Case 1: $\exists u : (u,y) \in F(x)$. Then, by construction, $(u,y) \in F(v)$.
Case 2: $\neg\exists u : (u,y) \in F(x)$. 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: $\exists z : (x,z) \in F(w) \vee (z,y) \in F(w)$. We consider two cases.
Case 1 $\exists z : (z,y) \in F(w)$. Then, by construction, $(z,y) \in F(v)$.
Case 2 $\neg\exists z : (z,y) \in F(w)$ and $\exists z : (x,z) \in F(w)$. 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) (line 7 – 9). Hence a vertex z' is chosen and $F(v) := F(v) \cup \{(x,z')\}$ is executed. □

Lemma 3.7
For all $(x,y) \in E : \exists v \in V : x,y \in X_v$.

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. We consider three cases:
Case 1: For each vertex $v \neq x,y$ on the path from x to y in $T : \exists z : (z,y) \in F(v)$. 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 $T : \exists z : (x,z) \in F(v)$. 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 with $\exists z : (z,y) \in F(v_1)$, and there exists a vertex $v_2 \neq x,y$ on the path from x to y in T with $\exists z : (z,y) \in F(v_2)$. 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 $\exists z : (z,w) \in X_x$, 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 $\{x \mid x \text{ is a predecessor of } w \text{ and } \exists z : (x,z) \in F(w)\} \supseteq \{x \mid x \text{ is a predecessor of } w \text{ and } \exists z : (x,z) \in F(v)\}$.
(iii) If x is a successor of w, and $\exists z : (w,z) \in X_x$, 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 $\{w \in V \mid v \in X_w\}$ forms a connected subtree of T.

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

Lemma 3.7 and 3.9 show that $\{X_v \mid v \in V\}, T)$ is indeed a correct 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 $dfs(v) \neq 1$, 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 $\exists z : (x,z) \in F(w) \vee (x,z) \in F(v)$. By lemma 3.8(ii), we have $\exists z : (x,z) \in F(w)$. Hence $|\{x \in X_v \mid x \text{ is a predecessor of } v\}| \leq |F(w)|+1 \leq c+1$.

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

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 lemma 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 l'th 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 l'th 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 or results that follow from theorems 3.11. and 3.12 or that show to what extent the techniques can be used.

Theorem 4.1
Let A be a minor-closed class of graphs, and suppose $GR_{2\times k} \notin A$ 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.) □

Theorem 4.2
Let H be a minor of the $2 \times k$ grid graph, and of the l'th 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. □

Theorem 4.3
Let H be a minor of the $2 \times k$ grid graph, and of the l'th 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 l'th circus graph. So we can apply theorem 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. 4.a)
- all cycles (fig. 4.b)
- all graphs that consists of two cycles that have exactly one edge in common (fig. 4.c)
- all caterpillars with hairs of length 1 (fig. 4.d)

- 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. 4.e).
- all subgraphs of these.

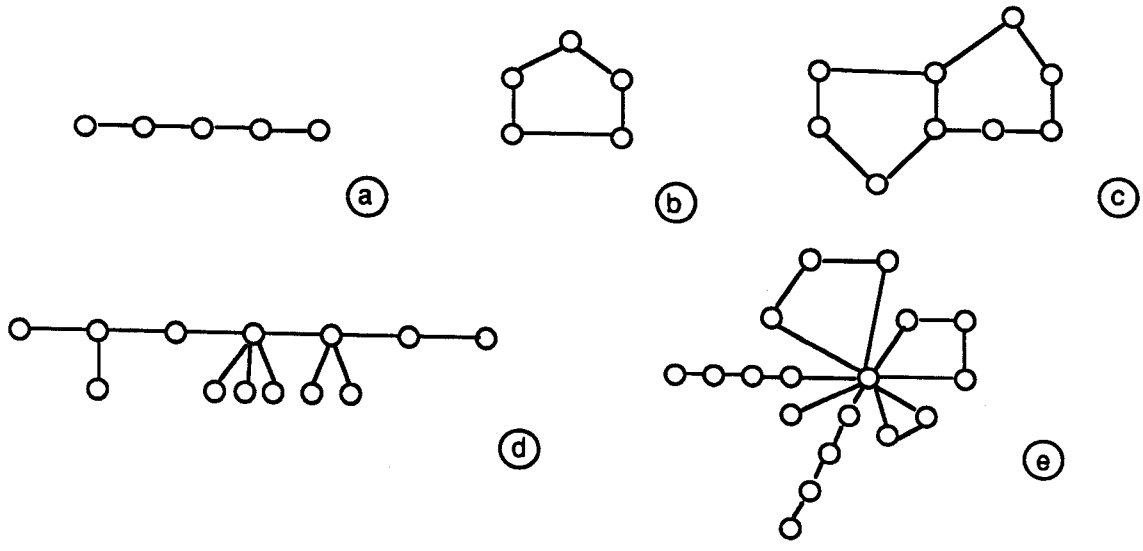

Figure 4: Graphs that are a minor of a $2 \times k$ grid and of a circus graph

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 can be tested in linear time. These results for paths and cycles were already obtained by Fellows and Langston [9] 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 [11]. Fellows and Langston [10] 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 contains $K_{1,k}$ as a minor: remove all non-tree edges from G, then contract over every edge between two non-leaf 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. □

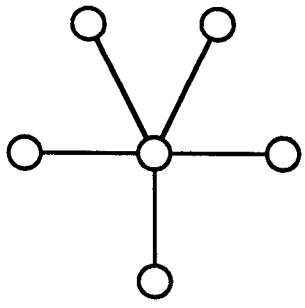

Figure 5: $K_{1,5}$

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., theorem 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 $v, \exists x : (w,x) \in E - F,\ x = v$ or x is a successor of $v\ \}$. 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 uses also $\mathcal{O}(n)$ time.

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). This result has already been obtained by Megiddo et al. [14] in a different 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.

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 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.

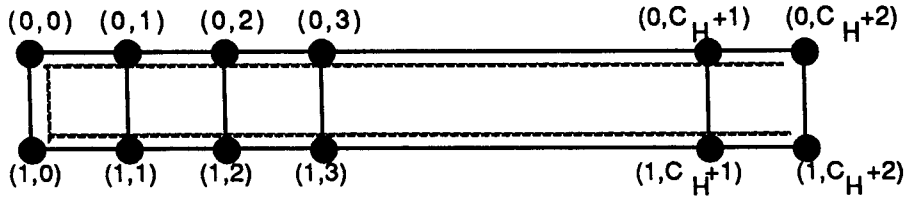

Figure 6: $GR_{2\times(C_H+3)}$ and T

Consider an arbitrary T-based tree-decomposition $(\{X_v \mid v \in V_G\}, T)$ 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 $\geq C_H + 1$, contradiction.
(ii) With a similar proof one can show that H is a minor of the $C_H + 3$ 'rd circus graph. Suppose not. Take $G = CC_{C_H+3}$, and use the depth-first-search spanning tree T, shown in fig. 7. (z_{C_H+3} is the root of T.)

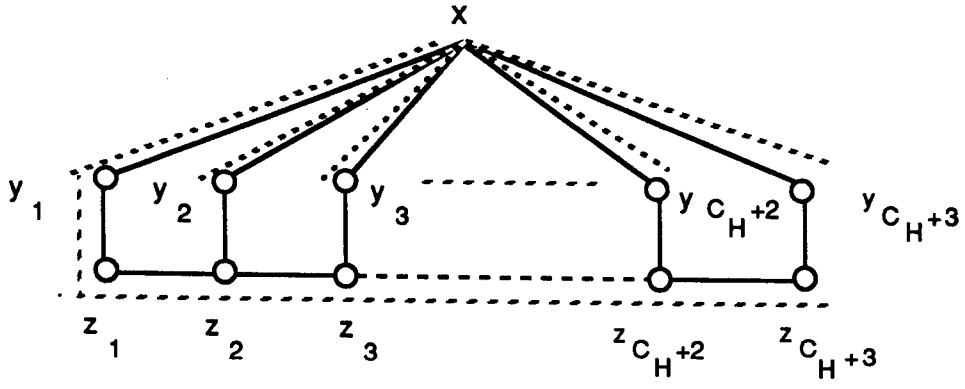

Figure 7: CC_{C_H+3} and T

From lemma 2.3. it follows that $|X_x| \geq C_H + 2$. □

Combining lemma 4.6. and theorem 3.11 gives

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 $GR_{2\times k}$ ($k \in \mathbb{Z}^+$), or A contains all graphs CC_l ($l \in \mathbb{Z}^+$). Then for every $m \in \mathbb{Z}^+$, 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 [9] 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 [8] they show that these cycles (paths) can be constructed by self-reduction algorithms, that use these decision algorithms as an oracle, in $\mathcal{O}(n^2 logn)$ ($\mathcal{O}(nlogn)$) 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 [16], 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 [16] 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 [9]. 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 in T (or all its predecessors). Then $(\{X_v \mid v \in V\}, T)$ 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 valves of k, and may be in some cases be a good alternative for branch-and bound algorithms or the algorithms of Monien [16].

We now comment on how to find the longest path between two specified vertices s, t. We only sketch the algorithm. First 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). Suppose 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 $\leq 2k-1$, similar as before, and then solve the problem (like in theorem 2.2) in $\mathcal{O}((2k)!2^{2k}n)$ time. If there exists a back edge between vertices with distance $\geq 2k-1$ in T, then G contains a cycle with length $\geq 2k$. One now can show that there exists a path from s to t with length $\geq k$ (using this cycle and the biconnectivity of G), and that this path can be found in $\mathcal{O}(e)$ time.

Theorem 5.2
(i) There exists an algorithm, that given a graph $G = (V, E)$, two vertices $s, t \in V$, an an integer k, either finds a simple path from s to t with length $\geq k$, or determines that such a path does not exists 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 graph 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 [12]. 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.

5.4 An interesting combinatorial problem, which arises with the longest cycle algorithm of Fellows and Langston (cf. section 5.1.) is the following: Given graph $G = (V, E)$, integer k, does G has a depth-first-search spanning tree $T = (V, F)$,

such that every back edge $(v, w) \in E - F$, the distance between v and w in T is at most k? The complexity of this problem seems to be open. Some related problems were proven to be NP-complete by Fellows et. al. [9].

Acknowledgements

Thanks are due to Mark de Berg en 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. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT*, 25:2–23, 1985.

[2] S. Arnborg, J. Lagergren, and D. Seese. Problems easy for tree-decomposable graphs (extended abstract). In *Proc. 15'th ICALP*, pages 38–51, Springer Verlag, Lect. Notes in Comp. Sc. 317, 1988.

[3] S. Arnborg and A. Proskurowski. *Linear time algorithms for NP-hard problems on graphs embedded in k-trees.* TRITA-NA-8404, Dept. of Num. Anal. and Comp. Sci., Royal Institute of Technology, Stockholm, Sweden, 1984.

[4] H. L. Bodlaender. NC-algorithms for graphs with small treewidth. In *Proc. Workshop on Graph-Theoretic Concepts in Computer Science WG'88*, pages 1–10, Springer Verlag, LNCS 344, 1988.

[5] B. Courcelle. *The monadic second-order logic of graphs I: Recognizable sets of finite graphs.* Technical Report I-8837, Dept. Comp. Sc, Univ. Bordeaux 1, 1988.

[6] B. Courcelle. The monadic second-order logic of graphs III: Treewidth, forbidden minors and complexity issues. 1988. Manuscript.

[7] P. Erdös and G. Szekeres. A combinatorial problem in geometry. *Compos. Math.*, 2:464–470, 1935.

[8] M. R. Fellows and M. A. Langston. *Fast Search Algorithms for Graph Layout Permutation Problems.* Technical Report CS-88-189, Dept. of Comp. Sc., Washington State Univ., 1988.

[9] M. R. Fellows and M. A. Langston. *On seach, decision and the efficiency of polynomial-time algorithms.* Technical Report CS-88-190, Dept. of Comp. Sc., Washington State Univ., 1988. To appear in Proc. STOC '89.