IMPROVED SELF-REDUCTION ALGORITHMS FOR GRAPHS WITH BOUNDED TREEWIDTH

Hans L. Bodlaender

RUU-CS-88-29
September 1988

Rijksuniversiteit Utrecht

Vakgroep informatica

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

IMPROVED SELF-REDUCTION ALGORITHMS FOR GRAPHS WITH BOUNDED TREEWIDTH

Hans L. Bodlaender

Technical Report RUU-CS-88-29
September 1988

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

ISSN: 0924–3275

IMPROVED SELF-REDUCTION ALGORITHMS FOR GRAPHS WITH BOUNDED TREEWIDTH

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

Abstract

Recent results of Robertson and Seymour show, that every class that is closed under taking of minors can be recognized in $\mathcal{O}(n^3)$ time. If there is a fixed upper bound on the treewidth of the graphs in the class, i.e. if there is a planar graph not in the class, then the class can be recognized in $\mathcal{O}(n^2)$ time. However, this result is non-constructive in two ways: the algorithm only decides on membership, but does not construct 'a solution', e.g. a linear ordering, decomposition or embedding; and no method is given to find the algorithms. In many cases, both non-constructive elements can be avoided, using techniques of Fellows and Langston, based on self-reduction. In this paper we introduce two techniques that help to reduce the running time of self-reduction algorithms. With help of these techniques we show that there exist $\mathcal{O}(n^2)$ algorithms, that decide on membership and construct solutions for treewidth, pathwidth, search number, vertex search number, cutwidth, modified cutwidth, vertex separation number, gate matrix layout, and progressive black-white pebbling, where in each case the parameter k is a fixed constant.

1 Introduction.

A graph G is said to be a minor of a graph H, if G can be obtained from a subgraph of H by a number of edge-contractions. (An edge-contraction is the operation that replaces two adjacent vertices v, w by a new vertex that is adjacent to all vertices, adjacent to v or w.) Robertson and Seymour [28] have shown that for every class of graphs F, that is closed under taking of minors, there is a finite set of graphs $ob(F)$, the obstruction set of F, such that for all graphs G: $G \in F$, if and only if there is no graph $H \in ob(F)$ that is a minor of G. Further, there is an $\mathcal{O}(n^3)$ algorithm for every fixed graph H, that tests whether H is a minor of a given graph G [27]. Thus, one can test membership in F in $\mathcal{O}(n^3)$ time. If the treewidth of graphs in F is bounded by some constant (or, equivalently, if there is at least one planar graph that is not in F [24]), then the minor-tests, and hence the membership in F-test can be done in $\mathcal{O}(n^2)$ time [27]). A similar characterization with an obstruction set exists for classes of graphs that are closed under immersions [25].

G is an immersion of H, if G can be obtained from a subgraph of H by a number of edge-lifts. An edge-lift is the operation that replaces edges (v, w) and (w, x) by an edge (v, x). For fixed H, one can test whether a given graph G contains H as an immersion in polynomial time, and in $\mathcal{O}(n^2)$ time if the treewidth of G is bounded. Many applications of these results were obtained by Fellows and Langston [13, 14, 15].

Note that these results are non-constructive in two ways: the algorithms only decide on membership in the class, but do not construct a solution like a linear ordering, decomposition or embedding, and no method is given to construct the algorithm: to write down this type of algorithm we must know the obstruction set of the class of graphs we want to recognize. However, in many cases, both non-constructive elements can be avoided with techniques of Brown, Fellows and Langston [10] and Fellows and Langston [16], based on self-reduction . We concentrate in this paper on the problem on finding solutions, for the case that there is a bound on the treewidth of the graphs.

Self-reduction is the technique to consult the decision algorithm a number of times with inputs derived from the original input, in order to construct the 'solution' to the problem. Algorithms of this type are also called 'oracle algorithms', and the decision algorithm is called the 'oracle'. The overhead of an oracle algorithm is the time, required for all operations, except those of the oracle, where each call to the oracle is counted as one unit of time.

This paper is organized as follows. In section 2 we review a number of definitions and results. In sections 3 and 4 we introduce two new techniques, that help to design faster constructive algorithms for immersion and minor closed classes of graphs with a fixed bound on the maximum treewidth. In section 5 we apply these techniques, and obtain $\mathcal{O}(n^2)$ algorithms that decide on membership and construct solutions for treewidth, pathwidth, search number, vertex search number, cutwidth, modified cutwidth, vertex separation number and gate matrix layout, where in each case the parameter k is a fixed constant. For each of these problems, except treewidth, algorithms with running time between $\mathcal{O}(n^3)$ and $\mathcal{O}(n^4)$ were designed by Fellows and Langston [12, 16]. Some final remarks are made in section 6.

2 Definitions and preliminary results.

In this section we give a number of well-known definitions and results. First we consider the important notion of treewidth, which was introduced by Robertson and Seymour [23].

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 all $i, j, k \in I$: if j lies on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

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 $\leq k$. See e.g. [1].

Very few NP-hard problems stay NP-hard, when we restrict them to a class of graphs with some fixed upper bound on the treewidth of the graphs in the class (see e.g. [3, 5, 7, 11, 18, 29]). In this paper we consider the approach of Courcelle [11] and Arnborg, Lagergren and Seese [3], as this approach appears to be most suitable for our purposes.

Courcelle [11] showed that every property that can be expressed in monadic second order form, can be tested in linear time for graphs that are given together with a tree-decomposition with constant bounded treewidth. Arnborg, Lagergren and Seese [3] extended the class of problems that can be dealt with. Consider logical formula's, that can use the following ingredients: the usual logical operations ($\wedge, \vee, \neg, \Rightarrow$, etc.), quantifications over vertices ($\exists v \in V, \forall v \in V$), edges ($\exists e \in E, \forall e \in E$), sets of vertices ($\exists W \subseteq V, \forall W \subseteq V$), and sets of edges ($\exists F \subseteq E, \forall F \subseteq E$), equality tests ($v = w, e = f, (v, w) = e$), membership tests ($v \in W, e \in F$), and incidence tests ($(v, w) \in E, (v, w) \in F$). The formula may be open, where the free variables are given interpretations as pre-specified vertices, edges, sets of vertices, or sets of edges, respectively. Properties, that are expressed in this way are called monadic second order graph properties. We use the following variant of the results of Courcelle [11] and Arnborg, Lagergren and Seese [3].

Theorem 2.1 [11, 3]
Let k be a constant. Let $\exists v \in V \Phi(G, v)$ $(\exists e \in E \Phi(G, e))$ be a monadic second order graph property. Then there exists a *linear time* algorithm, that given a graph $G = (V, E)$ together with a tree-decomposition of G with treewidth $\leq k$, either finds a vertex $v \in V$ such that $\Phi(G, v)$ (an edge $e \in E$, such that $\Phi(G, e)$), or decides that such a vertex v (edge e) does not exist.

For our purposes, it is important to note that for fixed graphs H, the properties "H is a minor of G", or "H is an immersion of G" can be expressed as monadic second order graph properties. In order to find tree-decomposition with small treewidth, we can use the following result, which is a direct corollary of results of Robertson and Seymour [26, 27].

Theorem 2.2 (Robertson, Seymour)
For every fixed $k \geq 1$, there is an $\mathcal{O}(n^2)$ algorithm that given a graph $G = (V, E)$, either decides that the treewidth of G is larger that k, or finds a tree-decomposition of G with treewidth $\leq 4\frac{1}{2}k$.

(Robertson and Seymour considered "branchwidth", and obtained a bound of $3k$.) Thus, for graphs with constant bounded treewidth, we can find in $\mathcal{O}(n^2)$ time a tree-decomposition with treewidth still bounded by a constant, although it does not have optimal treewidth.

Some other definitions we use:

$G[W]$	:	the subgraph of G, induced by W, i.e. $(W, \{(v,w) \in E \mid v, w \in W\})$
$G - \{v\}$	:	the subgraph of G, induced by $V - \{v\}$, $G[V - \{v\}]$.
$G - \{e\}$	:	the graph $(V, E - \{e\})$
clique $\{v_1, \ldots, v_k\}$	:	the complete graph on $\{v_1, \ldots, v_k\}$: $(\{v_1, \ldots, v_k\}, \{(v_i, v_j) \mid 1 \leq i, j \leq k, i \neq j\})$
$G \cup H$	:	the (not necessarily disjoint) union of G and H.

3 Quiet self-reductions.

In this section we propose the notion of "quiet" self-reduction . This rather simple idea is based on a closer observation of the $\mathcal{O}(n^2)$ minor test algorithm for graphs with bounded treewidth. Basically, this algorithm consists of two phases. In the first phase, the algorithm, indicated in theorem 2.2 is run. Either we decide that the treewidth of G is too large, and we know that G is a "no"-instance, or we find a tree-decomposition with treewidth $\mathcal{O}(1)$. This phase costs $\mathcal{O}(n^2)$ time. In the second phase, the tree-decomposition is used to perform the actual minor test, using dynamic programming, as in [3, 5, 7, 11, 29]. This second phase uses $\mathcal{O}(n)$ time.

An oracle algorithm may call this procedure a number of times, each time for a new graph G' that is obtained from modifications of the original input graph G. The number of such calls is usually at least $\mathcal{O}(n)$. Thus, it may be possible to save time, if we could be able to avoid the first phase for most of the calls to the minor-test algorithm. The following definition expresses a class of such oracle algorithms.

Definition.
An oracle algorithm is *quiet*, if, there are constants c_1, c_2, such that for any graph $G = (V, E)$, when the algorithm runs with G as input, then every graph $H = (W, F)$, that is input to the oracle, fulfills:

(i) $|W - (V \cap W)| \leq c_1$

(ii) $\exists W' \subseteq V : |W'| \leq c_2 \wedge ((v, w) \in F - (E \cap F) \Rightarrow v \in W' \vee w \in W')$

In other words, every graph that is input to the oracle, can be obtained by taking a subgraph of G, adding a constant number of new vertices, and for a constant number of vertices, adding a number of edges, starting at that vertex.

Theorem 3.1
Let k, l be constants. Let A be a quiet oracle algorithm, such that

(i) A yields the answer "no", if the treewidth of input graph G is larger than k.

(ii) A has overhead $f(n)$, and makes $g(n)$ oracle calls to an oracle $\mathcal{O}$.

(iii) A call to oracle $\mathcal{O}$ costs $\mathcal{O}(n)$ time, if the input-graph H to the oracle, is given together with a tree-decomposition of H with treewidth $\leq l$.

Then A can be implemented with an algorithm, that uses $\mathcal{O}(f(n) + n^2 + g(n) \cdot n)$ time.

Proof.
First run the algorithm of Robertson and Seymour, that either decides that treewidth $(G) \geq k$, or finds a tree-decomposition of G with constant treewidth. In the former case, output "no", and we are done. In the latter case, note that for input graph H to the oracle, one can find in $\mathcal{O}(n)$ time a tree-decomposition of H with constant treewidth: use the tree-decomposition of G, remove all vertices in $G - H$, and then add to each set X_i all vertices in $H - G$, and all vertices in the set W', defined by (v, w) edge in $H - G \Rightarrow v \in W' \vee \in W'$, $|W'|$ bounded by a constant. A tree-decomposition of H with constant bounded treewidth results. Hence, the total time for all oracle calls is bounded by $\mathcal{O}(g(n) \cdot n)$, and the total time for the algorithm is bounded by $\mathcal{O}(f(n) + n^2 + g(n) \cdot n)$. □

Theorem 3.1. can be applied to several problems, considered in [12]. In many cases, improvements with a factor up to $\mathcal{O}(n)$ can be made. However, we need a second technique in order to obtain $\mathcal{O}(n^2)$ algorithms.

4 Using monadic second order graph properties, instead of minor tests.

In this section we describe our second technique. It is based on the observation, that not only minor-tests, but also more complicated questions, if we write them as monadic second order graph properties, can be tested in linear time, given a tree-decomposition of G with constant bounded treewidth.

Lemma 4.1
Let $\varphi(G)$ be a monadic second order graph property. Each of the following properties can be expressed in monadic second order form:

(i) $\varphi(G - \{v\})$ $(G - \{v\} = (V - \{v\}, \{(w, x) | w \neq v, x \neq v\})$

(ii) $\varphi(G - \{e\})$ $(G - \{e\} = (V, E - \{e\}))$

(iii) $\varphi(G' = (V, E \cup \{(v, w)\}))$

(iv) $\varphi(G' = (V, E \cup \{(v, w) | w \in W\}))$

(v) $\varphi(G' = (V, E - \{(v, w) | w \in W\}))$

(v, w, e, W are free variables in the resulting formulas, but not in φ.)

Proof.
We can rewrite φ, inductively. For example, consider (i). We only consider a few cases, the other are similar. If $\varphi = \varphi_1 \vee \varphi_2$, then $\varphi(G - \{v\}) = \varphi_1(G - \{v\}) \vee \varphi_2(G - \{v\})$. If $\varphi = \exists W \in V \varphi_1(W, G)$, then $\varphi(G - \{v\}) = \exists w \in V : v \neq w \wedge \varphi_1(w, G - \{v\})$. If $\varphi = \exists W \subseteq V : \varphi_1(W, G)$, then $\varphi(G - \{v\}) = \exists W \subseteq V : \neg(v \in w) \wedge \varphi_1(W, G - \{v\})$. If $\varphi = (w = x)$, or $((w, x) \in F)$, then $\varphi(G - \{v\}) = \varphi$. The other cases are similar. □

This result can often be applied in the following way. Suppose we look for a vertex v (or edge e), such that G, with some local operations applied on v (or e) remains in a minor-closed class F. Lemma 4.1 shows that, if we can write these local operations in a suitable form, then we can find such a vertex v (or edge e), in linear time (supposing G is given with a constant width tree-decomposition). This can save up to a factor of $\mathcal{O}(n)$ time in comparison to algorithms that test for each vertex v the resulting modified graph G separately.

As a first example, consider the "k vertices within F" problem for a minor closed class of graphs F, with a fixed upper bound on the treewidth of graphs in F. (This problem, without assumptions on the treewidth was considered in [10].) I.e., we must find k vertices $v_1, \ldots, v_k$, such that $G - \{v_1, \ldots, v_k\} \in F$. One easily sees that if G is a "yes"-instance to this problem, then the treewidth of G is bounded by the maximum treewidth of a graph in F plus k.

Using the characterization with obstructions, the property $G \in F$ can be written as a monadic second order graph property, hence we can write $\exists v_1 \exists v_2 \ldots \exists v_k\ G - \{v_1, \ldots, v_k\} \in F$ as a monadic second order property. (Apply lemma 4.1(i) k times.) So suppose G is given together with a constant width tree-decomposition . Then we can find v_1 in linear time, using lemma 2.1. Using again lemma 2.1 on the property $\exists v_2 \ldots \exists v_k G - \{v_1, \ldots, v_k\} \in F$ we see that we can find v_2 in linear time. (v_1 is now a free variable with a predetermined value.) So, in k steps, each using $\mathcal{O}(n)$ time, we find $v_1, \ldots, v_k$, such that $G - \{v_1, \ldots, v_k\} \in F$, if they exist.

Theorem 4.2
Let k be a constant, and let F be a minor-closed class of graphs with a fixed upper bound on the treewidth of graphs in F. Then there exists an $\mathcal{O}(n)$ time algorithm, that given a graph $G = (V, E)$, together with a tree-decomposition of G with constant bounded treewidth , finds k vertices $v_1, \ldots, v_k$ (k edges $e_1, \ldots, e_k$), such that $G - \{v_1, \ldots, v_k\} \in F$ $(G - \{e_1, \ldots, e_k\} \in F)$, or decides that such collection of vertices (edges) does not exists.

5 Faster constructive algorithms for various problems.

5.1 Treewidth.

We now show that, for fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that constructs a tree-decomposition with treewidth $\leq k$, or decides that such a tree-decomposition does not exist for a given graph G. This improves on an $\mathcal{O}(n^{k+2})$ algorithm by Arnborg, Corneil and Proskurowski [2]. For $k = 1, 2, 3$ there exist linear time algorithms [4, 20]. For variable k, the problem is NP-complete [2].

Lemma 5.1 [6]
Let $G = (V, E)$ be a graph, with $W \subseteq V$ is a clique in G. Then, for any tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of G, there exists an $i \in I$ with $W \subseteq X_i$.

Our algorithm is based on the following lemma's, which are slight modifications of lemma's of Arnborg, Corneil and Proskurowski [2].

Lemma 5.2

(i) If $G = (V, E)$ has treewidth $\leq k$, and $|V| \geq k$, then there exist vertices $v_1, \ldots, v_k \in V$, with $G' = (V, E \cup \{(v_i, v_j) | 1 \leq i, j \leq k, i \neq j\}) = G \cup$ clique $(v_1, \ldots, v_k)$ has treewidth $\leq k$.

(ii) Suppose treewidth $(G) \leq k$, $v_1, \ldots, v_k$ form a clique in G. Let $V_1, \ldots, V_r$ be the sets of vertices of the connected components of $G[V - \{v_1, \ldots, v_k\}]$. Then, for all $i, 1 \leq i \leq r : G[V_i \cup \{v_1, \ldots, v_k\}]$ has treewidth $\leq k$.

(iii) Suppose treewidth $(G) \leq k, v_1, \ldots, v_k$ form a clique in G. Suppose $G[V - \{v_1, \ldots, v_k\}]$ is connected, $|V| \geq k + 2$. Then there exists a vertex $w \in V - \{v_1, \ldots, v_k\}$, such that $G' = (V, E \cup \{(v_i, w) | 1 \leq i \leq k\})$ has treewidth $\leq k$. Moreover, for each connected component V_i of $G[V - \{v_1, \ldots, v_k, w\}]$, there is at least one vertex $w_i \in \{v_1, \ldots, v_k, w\}$ with $\forall v \in V_i$ $(v, w_i) \neq E$, and $G'[V_i \cup (\{v_1, \ldots, v_k, w\} - \{w_i\}]$ has treewidth $\leq k$.

We now sketch our algorithm.

1. Run the "approximate tree-decomposition " algorithm of Robertson and Seymour (see lemma 2.2). If it tells us that the treewidth of G is larger than k, then output "no", and stop. Otherwise we have a tree-decomposition of G with treewidth $\mathcal{O}(1)$.

2. Test whether G contains a minor in the obstruction set of the graphs with treewidth $\leq k$. If so, then output "no", and stop. Otherwise, we know that treewidth $(G) \leq k$, and we continue with step 3. (Step 1 and 2 basically form the recognition algorithm of Robertson and Seymour [27]).

3. Find vertices $v_1, \ldots, v_k$ as indicated in lemma 5.2(i). This can be done in linear time, using lemma 2.1, observing that

   $$\exists v_1 \in V \exists v_2 \in V \cdots \exists v_k \in V : G' = (V, E \cup \{(v_i, v_j) | 1 \leq i, j \leq k, i \neq j\}) \mathit{hastreewidth} \leq k.$$

   can be written as a monadic second order graph property, by using lemma 4.1, and the characterization with forbidden minors. With k applications of lemma 2.1 we find vertices $v_1, v_2, \ldots, v_k$.

4. Determine the connected components $V_1, \ldots, V_r$ of $G[V - \{v_1, \ldots, v_k\}]$. Let $G' = G \cup \text{clique}\{v_1, \ldots, v_k\}$.

5. Now, for each $i, 1 \leq i \leq r$, find a tree-decomposition with treewidth $\leq k$ of $G'[V_i \cup \{v_1, \ldots, v_k\}]$, with a procedure, described below. Then build a tree-decomposition of G with treewidth $\leq k$ as follows:

   (a) Observe that, for each $i, 1 \leq i \leq r$, there must be a set $X_{\alpha(i)}$ in the tree-decomposition of $G'[V_i \cup \{v_i, \ldots, v_k\}]$ that contains $v_1, \ldots, v_k$, as these vertices form a clique in this graph.

   (b) Now take the disjoint union of all r tree-decompositions, add an extra set $X_o = \{v_1, \ldots, v_k\}$, and add a tree-edge from X_o to $X_{\alpha(i)}$ for all $i, 1 \leq i \leq r$. One can now check that a correct tree-decomposition of G with treewidth $\leq k$ results.

6. Next we describe a recursive procedure, that given a set $W \subseteq V$, with $G[W]$ connected, and vertices $v_1, \ldots, v_k \in V$, finds a tree-decomposition of $G' = G[W \cup \{v_1, \ldots, v_k\}] \cup \text{clique}(\{v_1, \ldots, v_k\})$, with treewidth $\leq k$. The set, associated to the root of the resulting tree contains $v_1, \ldots, v_k$.

   (a) If $|W| \leq 1$, then take the tree-decomposition $(\{X_1 = W \cup \{v_1, \ldots, v_k\}\}, (\{1\}, \emptyset))$.

   (b) Otherwise, find a vertex $w \in W$, with $G[W \cup \{v_1, \ldots, v_k\}] \cup$ clique $(\{v_1, \ldots, v_k, w\})$ has treewidth $\leq k$. This can be done in linear time, with the methods, exposed in sections 3 and 4. Lemma 5.2 guarantees us, that such a vertex w exists.

   (c) Determine the connected components $W_1, \ldots, W_r$, of $G[W - \{w\}]$.

   (d) For each of these connected components W_i, find $W_i \in \{v_1, \ldots, v_k, w\}$ with $\forall v \in W_i (v, w_i) \notin E$. (See lemma 5.2). Now call the procedure recursively with set W_i and vertices $\{v_1, \ldots, v_k, w\} - \{w_i\}$.

   (e) So now we have tree-decompositions of all graphs $G_i = G[W_i \cup (\{v_1, \ldots, v_k, w\} - \{w_i\})] \cup \text{clique}(\{v_1, \ldots, v_k, w\} - \{w_i\})$, with treewidth $\leq k$. The root of such a tree-decomposition contains $\{v_1, \ldots, v_k, w\} - \{w_i\}$. The desired tree-decomposition of G' can be built as follows: take the disjoint union of the tree-decomposition of G_i $(1 \leq i \leq r')$. Take a new set $X_r = \{v_1, \ldots, v_k, w\}$, which is taken as the root of the new tree-decompositions. Connect X_r to each of the roots of the tree-decomposition of graphs G_i. One can check that indeed the resulting structure is a tree-decomposition of G' with treewidth $\leq k$.

This completes the description of the algorithm. The time needed for steps 1,2,3,4 and 5 is bounded by $\mathcal{O}(n^2)$. Each call of the procedure in step 6 costs $\mathcal{O}(n)$ time, and this procedure is called $\mathcal{O}(n)$ times. Thus, the total time of our algorithm is $\mathcal{O}(n^2)$.

Theorem 5.3
For each constant k, there exists an $\mathcal{O}(n^2)$ algorithm, that for a given graph $G = (V, E)$ either decides that the treewidth of G is larger than k, or finds a tree-decomposition of G with treewidth $\leq k$.

(In [8] an (easier) $\mathcal{O}(n^3)$ algorithm is given.)

5.2 Search number and vertex search number.

In this section we consider the search number and vertex search number of a graph. A search strategy of a graph is a sequence of the following types of moves:

1. Place a searcher on a vertex.
2. Delete a searcher from a vertex.
3. Move a searcher over an edge.

All edges are initially *contaminated*. An edge (v, w) can become *cleared* by moving a searcher from v to w, while there is a second searcher on v, or all other edges, adjacent to v are already cleared. An edge can become *recontaminated*, when a move results in a path without searchers from a contaminated edge to the edge. The search number of G is the minimum number of searchers needed to clear all edges. It has been shown by LaPaugh [17] that for every graph G, there exists a search sequence, that uses the optimal number of searchers, and does not allow recontamination. Such a search sequence is called *progressive*. If we let vertices instead of edges be cleared or contaminated, then the vertex search number is the minimum number of searchers, needed to clear all vertices with a progressive search sequence. Determining the minimum number of searchers needed is NP-complete [21]. Note that for a progressive search strategy that clears all vertices, we may assume that never two searchers occupy the same vertex, and that once a searcher has left a vertex, then no searcher will visit that vertex again. Note that, for fixed k, the classes of graphs with search number of vertex search number $\leq k$ are closed under taking of minors. Also, if the (vertex) search number of G is k, then treewidth $(G) \leq k$ (see e.g. [9]).

Definition.
$BW(n,k) = (\{v_1, \ldots, v_n\}, \{(v_i, v_j) | 1 \leq i, j \leq n, |i-j| \leq k\})$.

($BW(n,k)$ is the maximal graph on n vertices with bandwidth k.)

Lemma 5.4

(i) If there exists a progressive search strategy that clears all vertices of $G = (V, E)$ with k searchers, then there exists one with the first k moves the placing of a searcher ($|V| \geq k$).

(ii) There exists a progressive search strategy that clears all vertices of G, and that starts with placing a searcher on vertices $w_1, \ldots, w_k$, if and only if the graph G', obtained by taking the disjoint union of G and $BW(2k+1, k)$ and then identifying vertices v_i and w_i for $1 \leq i \leq k$, has vertex search number $\leq k$.

Proof.
(i) One can obtain the desired search strategy by first executing the first k moves of the type "place a searcher on a vertex", and then executing all other moves in sequence.
(ii) $\Rightarrow$ Use the following search strategy: place searchers on vertices $v_{k+2}, v_{k+3}, \ldots, v_{2k+1}$. Then move a searcher from v_i to v_{i-k}, for $i = 2k+1, 2k, 2k-1, \ldots, k+1$. Now we have searchers on $w_1 = v_1, \ldots, w_k = v_k$. Continue with the given search strategy that clears all vertices of G and starts with searchers on $w_1, \ldots, w_k$.
$\Leftarrow$ Consider a progressive search strategy that clears all vertices of G' with k searchers. Consider the first move that removes a searcher from a vertex $v_i \in \{v_{k+1}, \ldots, v_{2k+1}\}$ or moves a searcher away from a vertex $v_i \in \{v_{k+1}, \ldots, v_{2k+1}\}$. It follows that after this move, all vertices in $\{v_{k+1}, \ldots, v_{2k+1}\} - \{v_i\}$ must contain a searcher. Also, vertices $v_{i-1}, \ldots, v_{i-k}$ must be cleared or contain a searcher. It follows that either all vertices in G are cleared or uncleared. Suppose the latter. It follows that $i = 2k+1$, and that the next $k+1$ moves are: move a searcher from v_j to v_{j-k}, for $j = 2k+1, \ldots, k+1$. We then have searchers on $v_1 = w_1, \ldots, v_k = w_k$. The remaining search sequence is a progressive search sequence that clears all vertices of G, starting with searchers on $v_1, \ldots, v_k$. In the case that all vertices in G are cleared after moving the searcher from v_i, we use the same argument for the search strategy, that is obtained by "reversing" the original strategy. $\square$

Lemma 5.4 gives us a method to find the first k vertices where a searcher is placed: we must find vertices $v_1, \ldots, v_k$, such that the graph $(V \cup \{w_{k+1}, \ldots, w_{2k+1}\}, E \cup \{(v_i, v_j) | 1 \leq i, j \leq k, i \neq j\} \cup \{(v_i, w_j) | 1 \leq i \leq k < j \leq i+k\} \cup \{(w_i, w_j) | k \leq i, j \leq 2k+1, 0 < |i-j| \leq k\})$ has vertex search number k. This can be done in linear time, by combining the techniques of section 3 and 4. (Add $w_{k+1}, \ldots, w_{2k+1}$ to G, and all edges between them. Construct a constant width tree-decomposition of the resulting graph G'. Write: "$\exists v_1 \ldots \exists v_k$: if we add edges between $v_i, v_j, 1 \leq i \leq j \leq k$, and v_i, w_j with $j - i \leq k$, then the resulting graph has vertex search number $\leq k$" as a monadic second order graph property, using the characterization with forbidden minors, and lemma 4.1.).

In order to find the next move, we distinguish between two cases.

Case 1: There exists a vertex v_i, containing a searcher, that is not adjacent to a vertex that does not contain a searcher. Then one may assume that the next move in our search strategy is the removal of the searcher from v_i. Now remove v_i from G, and find, recursively a progressive search strategy that clears all vertices of $G - \{v\}$ with k searchers, that starts with placing a vertex on each of the vertices $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k$. (This can be done, similar as how we found the first k moves. Only let $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k$ be free variables in the monadic second order graph property that is tested.) The resulting search sequence can easily be extended to the desired search sequence.

Case 2: Such a vertex v_i does not exists. Then necessarily the next move must be the moving of a searcher from a vertex v_i to an adjacent uncleared vertex. Note that for each searcher, there is at most one vertex where it can move to, otherwise the vertex it leaves will become recontaminated. So we have to consider at most k possible moves. For each possible move from v_i to w, test whether there exists a progressive search strategy that clears the vertices of $G - \{v_i\}$ with k searchers, and that starts with placing a searcher on $v_1, \ldots, v_{i-1}, v_{i+1}, v_k$, and w. This can be tested, similar as above, in linear time. Do this for each of the k possible moves, until a good move is found. Then find recursively a progressive search strategy, that clears the vertices of G with k searchers, that starts with placing a searcher on $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k$ and w.

As in total $\mathcal{O}(n)$ moves are made, the total time of the algorithm is $\mathcal{O}(n^2)$.

Theorem 5.5
For each constant k, there exists an $\mathcal{O}(n^2)$ algorithm, that for a given graph $G = (V, E)$, either decides that the vertex search number of G is larger than k, or finds a progressive search sequence that clears the vertices of G with k searchers.

We can use this result to obtain the following theorem as an easy corollary.

Theorem 5.6
For each constant k, there exists an $\mathcal{O}(n^2)$ algorithm, that for a given graph $G = (v, E)$ either decides that the search number of G is larger than k, or finds a progressive search strategy that clears all edges of G with k searchers.

Proof.
Let G' be the graph, obtained by subdividing each edge in G once, i.e. $G' = (V \cup E, \{(v, e) | v \in V, e \in E, \exists w \in V : (v, w) = e\})$. The vertex search number of G' equals the search number of G, and the corresponding search strategies can be easily transformed into each other. □

5.3 Other problems.

Several other problems can be dealt with in the same manner. The techniques are similar to those used for the vertex search number problem, but the details are different.

We give the definitions of the problems, that are considered. A linear ordering of a graph $G = (V, E)$ is a bijection $V \to \{1, 2, \ldots, |V|\}$. The cutwidth of a linear ordering f is $\max_{1 \le i < n} |\{(v, w) \in E | f(v) \le i < f(w)\}|$. The cutwidth of G is the minimum cutwidth over all linear orderings of G. The vertex separation of a linear ordering f is $\max_{1 \le i < n} |\{v \in V | f(v) \le i \wedge \exists w \in V : (v, w) \in E \cap f(w) > i\}|$. The vertex separation number of G is the minimum vertex separation over all linear orderings of G. The modified cutwidth of a linear ordering f is $\max_{1 \le i < n} |\{(v, w) \in E | f(v) < i < f(w)\}|$. The modified cutwidth of G is the minimum modified cutwidth over all linear orderings of G. A path-decomposition of G is a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of G, where T is a path, i.e. T is a tree with every node degree 1 or 2. The pathwidth of path-decomposition $(\{X_i | i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The pathwidth of G is the minimum pathwidth over all path-decompositions of G.

Determining the cutwidth, modified cutwidth, vertex separation number or pathwidth of a graph is NP-complete (see e.g. [2, 22]). For fixed k, Fellows and Langston obtained $\mathcal{O}(n^3)$ and $\mathcal{O}(n^3 \log n)$ algorithms [12, 16].

Theorem 5.7

(i) For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that for a given graph $G = (V, E)$ either decides that the cutwidth of G is larger than k, or finds a linear ordering of G with cutwidth $\le k$.

(ii) For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that for a given graph $G = (V, E)$ either decides that the pathwidth of G is larger than k, or finds a path-decomposition of G with pathwidth $\le k$.

(iii) For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that for a given graph $G = (V, E)$ either decides that the vertex separation number of G is larger than k, or finds a linear ordering of G with vertex separation $\le k$.

(iv) For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm either decides that for a given graph $G = (V, E)$, the modified cutwidth of G is larger than k, or finds a linear ordering of G with modified cutwidth $\le k$.

Proof.

(i) Use the self-reduction of Brown, Fellows and Langston [10], and apply the techniques of section 3 and 4. Parallel edges can be avoided by putting a vertex on the middle of these edges.

(ii), (iii), (iv) Omitted. □

Corollary 5.8
For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm, that for a given Boolean matrix M, either finds a column permutation of M such that if in each row every 0 lying between the row's leftmost and rightmost 1 is changed to a *, then no column contains more than k 1's and *s, or decides that such a column permutation does not exist.

Proof.
There is a linear transformation from this problem to the problem of finding path-decompositions with pathwidth $\leq k - 1$ [14, 16]. □

Corollary 5.9
For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that, for a given directed acyclic graph $G = (V, E)$, either gives a pebbling strategy for the progressive black-white pebbling game on G, that uses k pebbles (see [19] for a definition) or decides that such pebbling strategy does not exist.

Proof.
There is a linear transformation from this problem to the vertex separation problem and vice versa [19]. □

6 Final remarks.

As discussed in section 1, the algorithms are non-constructive in the sense that one knows that an algorithm exists, but we do not have a concrete algorithm of which we know that it is correct. In order to overcome this problem, Fellows and Langston [12] designed a technique that allows us to actually construct algorithms without knowing the exact obstruction set. The technique does not work always, but only if we have an (efficient) oracle algorithm, producing "solutions", an (efficient) algorithm that checks whether a "candidate-solution" is a correct solution, and an arbitrary (other) algorithm, that decides on membership in the considered class of graphs. The resulting algorithm has the following form. Let T be a minor or immersion closed class of graphs.

1. Let S be a subset of the obstruction set of T.
2. Check whether the input graph G has a graph in S as a minor. If so, output "no" ($G \neq T$), and stop.
3. Use the oracle algorithm to produce a 'solution', using S as an obstruction set.

4. Check the produced solution. If it is a correct solution, output it ($G \in T$), and stop.

5. (S is not the complete obstruction set of T). Enumerate all graphs, until a graph $G, G \notin S, G \notin T$, all minors of G are in T, are found. (G is a new element of the obstruction set). Put G in S. Go to step 2.

We can apply this technique to each of the problems considered in section 5. Observe that the method of Arnborg, Lagergren and Seese [3] that obtains a linear time algorithm on bounded treewidth graphs, given an extended monadic second order graph property, is in fact an automatic procedure. As our obstruction sets may grow, our monadic second order graph properties can vary during this algorithm, so we need to implement this procedure in order to combine the techniques, and obtain fully constructive $\mathcal{O}(n^2)$ algorithms for the problems considered in section 5.

References

[1] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - A survey. *BIT*, 25:2–23, 1985.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[3] 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.

[4] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *Siam J. Alg. Disc. Meth.*, 7:305–314, 1986.

[5] 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.

[6] H. L. Bodlaender. *Dynamic programming algorithms on graphs with bounded tree-width.* Tech. Rep., Lab. for Comp. Science, M.I.T., 1987. Ext. abstract in proceedings ICALP 88.

[7] H. L. Bodlaender. *NC-algorithms for graphs with small treewidth.* Technical Report RUU-CS-88-4, Dept. of Comp. Science, Univ. of Utrecht, Utrecht, 1988. To appear in: Proc. Workshop on Graph-Theoretical Concepts in Computer Science, 1988.

[8] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. In *Proc. 1st Scandinavian Workshop on Algorithm Theory*, pages 223–232, 1988.

[9] H. L. Bodlaender. Some classes of graphs with bounded treewidth. *Bulletin of the EATCS*, 1988. To appear.

[10] D. Brown, M. Fellows, and M. Langston. Nonconstructive polynomial-time decidability and self-reducibility. In *Princeton Forum on Algorithms and Complexity*, 1987.

[11] 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.

[12] M. R. Fellows and M. A. Langston. Fast self-reduction algorithms for combinatorial problems of VLSI design. In *Proc. 3rd Aegean Workshop on Computing*, 1988.

[13] M. R. Fellows and M. A. Langston. Layout permutation problems and well-partially-ordered sets. In *5th MIT Conf. on Advanced Research in VLSI*, pages 315–327, 1988.

[14] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Inform. Proc. Letters*, 26:157–162, 1987.

[15] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. 1987. To appear in JACM.

[16] M. R. Fellows and M. A. Langston. On seach, decision and the efficiency of polynomial-time algorithms. 1988. Extended abstract.

[17] A. S. LaPaugh. *Recontamination Does Not Help to Search a Graph.* Technical Report, Comp. Sci. Dept, Princeton Univ., New Yersey, 1982.

[18] C. Lautemann. Efficient algorithms on context-free graph languages. In *Proc. 15'th ICALP*, pages 362–378, Springer Verlag, Lect. Notes in Comp. Sc. 317, 1988.

[19] T. Lengauer. Black-white pebbles and graph separation. *Acta Inf.*, 16:465–475, 1981.

[20] J. Matousěk and R. Thomas. Algorithms finding tree-decompositions of graphs. 1988. Unpublished paper.

[21] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35:18–44, 1988.

[22] B. Monien and I. H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theor. Comp. Sc.*, 58:209–229, 1988.

[23] N. Robertson and P. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. of Algorithms*, 7:309–322, 1986.

[24] N. Robertson and P. Seymour. Graph minors. III. Planar tree-width. *J. Combin. Theory Series B*, 36:49–64, 1984.

[25] N. Robertson and P. Seymour. Graph minors. IV. Tree-width and well-quadi-ordering. Manuscript.

[26] N. Robertson and P. Seymour. Graph minors. X. Obstructions to tree-decompositions. 1986. Manuscript.

[27] N. Robertson and P. Seymour. Graph minors. XIII. The disjoint paths problem. 1986. Manuscript.

[28] N. Robertson and P. Seymour. Graph minors. XVI. Wagner's conjecture. To appear.

[29] P. Scheffler. *Linear-time algorithms for NP-complete problems restricted to partial k-trees.* Report R-MATH-03/87, Karl-Weierstrass-Institut Für Mathematik, Berlin, GDR, 1987.

IMPROVED SELF-REDUCTION ALGORITHMS FOR GRAPHS WITH BOUNDED TREEWIDTH

Hans L. Bodlaender

RUU-CS-88-29
September 1988

IMPROVED SELF-REDUCTION ALGORITHMS FOR GRAPHS WITH BOUNDED TREEWIDTH

Hans L. Bodlaender

Technical Report RUU-CS-88-29
September 1988

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

ISSN: 0924–3275

IMPROVED SELF-REDUCTION ALGORITHMS FOR GRAPHS WITH BOUNDED TREEWIDTH

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

Abstract

Recent results of Robertson and Seymour show, that every class that is closed under taking of minors can be recognized in $\mathcal{O}(n^3)$ time. If there is a fixed upper bound on the treewidth of the graphs in the class, i.e. if there is a planar graph not in the class, then the class can be recognized in $\mathcal{O}(n^2)$ time. However, this result is non-constructive in two ways: the algorithm only decides on membership, but does not construct 'a solution', e.g. a linear ordering, decomposition or embedding; and no method is given to find the algorithms. In many cases, both non-constructive elements can be avoided, using techniques of Fellows and Langston, based on self-reduction. In this paper we introduce two techniques that help to reduce the running time of self-reduction algorithms. With help of these techniques we show that there exist $\mathcal{O}(n^2)$ algorithms, that decide on membership and construct solutions for treewidth, pathwidth, search number, vertex search number, cutwidth, modified cutwidth, vertex separation number, gate matrix layout, and progressive black-white pebbling, where in each case the parameter k is a fixed constant.

1 Introduction.

A graph G is said to be a minor of a graph H, if G can be obtained from a subgraph of H by a number of edge-contractions. (An edge-contraction is the operation that replaces two adjacent vertices v, w by a new vertex that is adjacent to all vertices, adjacent to v or w.) Robertson and Seymour [28] have shown that for every class of graphs F, that is closed under taking of minors, there is a finite set of graphs $ob(F)$, the obstruction set of F, such that for all graphs G: $G \in F$, if and only if there is no graph $H \in ob(F)$ that is a minor of G. Further, there is an $\mathcal{O}(n^3)$ algorithm for every fixed graph H, that tests whether H is a minor of a given graph G [27]. Thus, one can test membership in F in $\mathcal{O}(n^3)$ time. If the treewidth of graphs in F is bounded by some constant (or, equivalently, if there is at least one planar graph that is not in F [24]), then the minor-tests, and hence the membership in F-test can be done in $\mathcal{O}(n^2)$ time [27]). A similar characterization with an obstruction set exists for classes of graphs that are closed under immersions [25].

G is an immersion of H, if G can be obtained from a subgraph of H by a number of edge-lifts. An edge-lift is the operation that replaces edges (v, w) and (w, x) by an edge (v, x). For fixed H, one can test whether a given graph G contains H as an immersion in polynomial time, and in $\mathcal{O}(n^2)$ time if the treewidth of G is bounded. Many applications of these results were obtained by Fellows and Langston [13, 14, 15].

Note that these results are non-constructive in two ways: the algorithms only decide on membership in the class, but do not construct a solution like a linear ordering, decomposition or embedding, and no method is given to construct the algorithm: to write down this type of algorithm we must know the obstruction set of the class of graphs we want to recognize. However, in many cases, both non-constructive elements can be avoided with techniques of Brown, Fellows and Langston [10] and Fellows and Langston [16], based on self-reduction . We concentrate in this paper on the problem on finding solutions, for the case that there is a bound on the treewidth of the graphs.

Self-reduction is the technique to consult the decision algorithm a number of times with inputs derived from the original input, in order to construct the 'solution' to the problem. Algorithms of this type are also called 'oracle algorithms', and the decision algorithm is called the 'oracle'. The overhead of an oracle algorithm is the time, required for all operations, except those of the oracle, where each call to the oracle is counted as one unit of time.

This paper is organized as follows. In section 2 we review a number of definitions and results. In sections 3 and 4 we introduce two new techniques, that help to design faster constructive algorithms for immersion and minor closed classes of graphs with a fixed bound on the maximum treewidth. In section 5 we apply these techniques, and obtain $\mathcal{O}(n^2)$ algorithms that decide on membership and construct solutions for treewidth, pathwidth, search number, vertex search number, cutwidth, modified cutwidth, vertex separation number and gate matrix layout, where in each case the parameter k is a fixed constant. For each of these problems, except treewidth, algorithms with running time between $\mathcal{O}(n^3)$ and $\mathcal{O}(n^4)$ were designed by Fellows and Langston [12, 16]. Some final remarks are made in section 6.

2 Definitions and preliminary results.

In this section we give a number of well-known definitions and results. First we consider the important notion of treewidth, which was introduced by Robertson and Seymour [23].

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 all $i, j, k \in I$: if j lies on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

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 $\leq k$. See e.g. [1].

Very few NP-hard problems stay NP-hard, when we restrict them to a class of graphs with some fixed upper bound on the treewidth of the graphs in the class (see e.g. [3, 5, 7, 11, 18, 29]). In this paper we consider the approach of Courcelle [11] and Arnborg, Lagergren and Seese [3], as this approach appears to be most suitable for our purposes.

Courcelle [11] showed that every property that can be expressed in monadic second order form, can be tested in linear time for graphs that are given together with a tree-decomposition with constant bounded treewidth. Arnborg, Lagergren and Seese [3] extended the class of problems that can be dealt with. Consider logical formula's, that can use the following ingredients: the usual logical operations ($\wedge, \vee, \neg, \Rightarrow$, etc.), quantifications over vertices ($\exists v \in V, \forall v \in V$), edges ($\exists e \in E, \forall e \in E$), sets of vertices ($\exists W \subseteq V, \forall W \subseteq V$), and sets of edges ($\exists F \subseteq E, \forall F \subseteq E$), equality tests ($v = w, e = f, (v, w) = e$), membership tests ($v \in W, e \in F$), and incidence tests ($(v, w) \in E, (v, w) \in F$). The formula may be open, where the free variables are given interpretations as pre-specified vertices, edges, sets of vertices, or sets of edges, respectively. Properties, that are expressed in this way are called monadic second order graph properties. We use the following variant of the results of Courcelle [11] and Arnborg, Lagergren and Seese [3].

Theorem 2.1 [11, 3]
Let k be a constant. Let $\exists v \in V \Phi(G, v)$ $(\exists e \in E \Phi(G, e))$ be a monadic second order graph property. Then there exists a *linear time* algorithm, that given a graph $G = (V, E)$ together with a tree-decomposition of G with treewidth $\leq k$, either finds a vertex $v \in V$ such that $\Phi(G, v)$ (an edge $e \in E$, such that $\Phi(G, e)$), or decides that such a vertex v (edge e) does not exist.

For our purposes, it is important to note that for fixed graphs H, the properties "H is a minor of G", or "H is an immersion of G" can be expressed as monadic second order graph properties. In order to find tree-decomposition with small treewidth, we can use the following result, which is a direct corollary of results of Robertson and Seymour [26, 27].

Theorem 2.2 (Robertson, Seymour)
For every fixed $k \geq 1$, there is an $\mathcal{O}(n^2)$ algorithm that given a graph $G = (V, E)$, either decides that the treewidth of G is larger that k, or finds a tree-decomposition of G with treewidth $\leq 4\frac{1}{2}k$.

(Robertson and Seymour considered "branchwidth", and obtained a bound of $3k$.) Thus, for graphs with constant bounded treewidth, we can find in $\mathcal{O}(n^2)$ time a tree-decomposition with treewidth still bounded by a constant, although it does not have optimal treewidth.

Some other definitions we use:

$G[W]$	:	the subgraph of G, induced by W, i.e. $(W, \{(v,w) \in E \mid v, w \in W\})$
$G - \{v\}$	:	the subgraph of G, induced by $V - \{v\}$, $G[V - \{v\}]$.
$G - \{e\}$	:	the graph $(V, E - \{e\})$
clique $\{v_1, \ldots, v_k\}$	:	the complete graph on $\{v_1, \ldots, v_k\}$: $(\{v_1, \ldots, v_k\}, \{(v_i, v_j) \mid 1 \leq i, j \leq k, i \neq j\})$
$G \cup H$	:	the (not necessarily disjoint) union of G and H.

3 Quiet self-reductions.

In this section we propose the notion of "quiet" self-reduction . This rather simple idea is based on a closer observation of the $\mathcal{O}(n^2)$ minor test algorithm for graphs with bounded treewidth. Basically, this algorithm consists of two phases. In the first phase, the algorithm, indicated in theorem 2.2 is run. Either we decide that the treewidth of G is too large, and we know that G is a "no"-instance, or we find a tree-decomposition with treewidth $\mathcal{O}(1)$. This phase costs $\mathcal{O}(n^2)$ time. In the second phase, the tree-decomposition is used to perform the actual minor test, using dynamic programming, as in [3, 5, 7, 11, 29]. This second phase uses $\mathcal{O}(n)$ time.

An oracle algorithm may call this procedure a number of times, each time for a new graph G' that is obtained from modifications of the original input graph G. The number of such calls is usually at least $\mathcal{O}(n)$. Thus, it may be possible to save time, if we could be able to avoid the first phase for most of the calls to the minor-test algorithm. The following definition expresses a class of such oracle algorithms.

Definition.
An oracle algorithm is *quiet*, if, there are constants c_1, c_2, such that for any graph $G = (V, E)$, when the algorithm runs with G as input, then every graph $H = (W, F)$, that is input to the oracle, fulfills:

(i) $|W - (V \cap W)| \leq c_1$

(ii) $\exists W' \subseteq V : |W'| \leq c_2 \wedge ((v,w) \in F - (E \cap F) \Rightarrow v \in W' \vee w \in W')$

In other words, every graph that is input to the oracle, can be obtained by taking a subgraph of G, adding a constant number of new vertices, and for a constant number of vertices, adding a number of edges, starting at that vertex.

Theorem 3.1
Let k, l be constants. Let A be a quiet oracle algorithm, such that

(i) A yields the answer "no", if the treewidth of input graph G is larger than k.

(ii) A has overhead $f(n)$, and makes $g(n)$ oracle calls to an oracle $\mathcal{O}$.

(iii) A call to oracle $\mathcal{O}$ costs $\mathcal{O}(n)$ time, if the input-graph H to the oracle, is given together with a tree-decomposition of H with treewidth $\leq l$.

Then A can be implemented with an algorithm, that uses $\mathcal{O}(f(n) + n^2 + g(n) \cdot n)$ time.

Proof.
First run the algorithm of Robertson and Seymour, that either decides that treewidth $(G) \geq k$, or finds a tree-decomposition of G with constant treewidth. In the former case, output "no", and we are done. In the latter case, note that for input graph H to the oracle, one can find in $\mathcal{O}(n)$ time a tree-decomposition of H with constant treewidth: use the tree-decomposition of G, remove all vertices in $G - H$, and then add to each set X_i all vertices in $H - G$, and all vertices in the set W', defined by (v, w) edge in $H - G \Rightarrow v \in W' \vee \in W'$, $|W'|$ bounded by a constant. A tree-decomposition of H with constant bounded treewidth results. Hence, the total time for all oracle calls is bounded by $\mathcal{O}(g(n) \cdot n)$, and the total time for the algorithm is bounded by $\mathcal{O}(f(n) + n^2 + g(n) \cdot n)$. □

Theorem 3.1. can be applied to several problems, considered in [12]. In many cases, improvements with a factor up to $\mathcal{O}(n)$ can be made. However, we need a second technique in order to obtain $\mathcal{O}(n^2)$ algorithms.

4 Using monadic second order graph properties, instead of minor tests.

In this section we describe our second technique. It is based on the observation, that not only minor-tests, but also more complicated questions, if we write them as monadic second order graph properties, can be tested in linear time, given a tree-decomposition of G with constant bounded treewidth.

Lemma 4.1
Let $\varphi(G)$ be a monadic second order graph property. Each of the following properties can be expressed in monadic second order form:

(i) $\varphi(G - \{v\})$ $(G - \{v\} = (V - \{v\}, \{(w, x) | w \neq v, x \neq v\})$

(ii) $\varphi(G-\{e\})\quad (G-\{e\}=(V,E-\{e\}))$

(iii) $\varphi(G'=(V,E\cup\{(v,w)\}))$

(iv) $\varphi(G'=(V,E\cup\{(v,w)|w\in W\}))$

(v) $\varphi(G'=(V,E-\{(v,w)|w\in W\}))$

(v, w, e, W are free variables in the resulting formulas, but not in φ.)

Proof.
We can rewrite φ, inductively. For example, consider (i). We only consider a few cases, the other are similar. If $\varphi=\varphi_1\vee\varphi_2$, then $\varphi(G-\{v\})=\varphi_1(G-\{v\})\vee\varphi_2(G-\{v\})$. If $\varphi=\exists W\in V\varphi_1(W,G)$, then $\varphi(G-\{v\})=\exists w\in V: v\neq w\wedge\varphi_1(w,G-\{v\})$. If $\varphi=\exists W\subseteq V:\varphi_1(W,G)$, then $\varphi(G-\{v\})=\exists W\subseteq V:\neg(v\in w)\wedge\varphi_1(W,G-\{v\})$. If $\varphi=(w=x)$, or $((w,x)\in F)$, then $\varphi(G-\{v\})=\varphi$. The other cases are similar.
□

This result can often be applied in the following way. Suppose we look for a vertex v (or edge e), such that G, with some local operations applied on v (or e) remains in a minor-closed class F. Lemma 4.1 shows that, if we can write these local operations in a suitable form, then we can find such a vertex v (or edge e), in linear time (supposing G is given with a constant width tree-decomposition). This can save up to a factor of $\mathcal{O}(n)$ time in comparison to algorithms that test for each vertex v the resulting modified graph G separately.

As a first example, consider the "k vertices within F" problem for a minor closed class of graphs F, with a fixed upper bound on the treewidth of graphs in F. (This problem, without assumptions on the treewidth was considered in [10].) I.e., we must find k vertices $v_1,\ldots,v_k$, such that $G-\{v_1,\ldots,v_k\}\in F$. One easily sees that if G is a "yes"-instance to this problem, then the treewidth of G is bounded by the maximum treewidth of a graph in F plus k.

Using the characterization with obstructions, the property $G\in F$ can be written as a monadic second order graph property, hence we can write $\exists v_1\exists v_2\ldots\exists v_k\ G-\{v_1,\ldots,v_k\}\in F$ as a monadic second order property. (Apply lemma 4.1(i) k times.) So suppose G is given together with a constant width tree-decomposition . Then we can find v_1 in linear time, using lemma 2.1. Using again lemma 2.1 on the property $\exists v_2\ldots\exists v_k G-\{v_1,\ldots,v_k\}\in F$ we see that we can find v_2 in linear time. (v_1 is now a free variable with a predetermined value.) So, in k steps, each using $\mathcal{O}(n)$ time, we find $v_1,\ldots,v_k$, such that $G-\{v_1,\ldots,v_k\}\in F$, if they exist.

Theorem 4.2
Let k be a constant, and let F be a minor-closed class of graphs with a fixed upper bound on the treewidth of graphs in F. Then there exists an $\mathcal{O}(n)$ time algorithm, that given a graph $G=(V,E)$, together with a tree-decomposition of G with constant bounded treewidth , finds k vertices $v_1,\ldots,v_k$ (k edges $e_1,\ldots,e_k$), such that $G-\{v_1,\ldots,v_k\}\in F$ ($G-\{e_1,\ldots,e_k\}\in F$), or decides that such collection of vertices (edges) does not exists.

5 Faster constructive algorithms for various problems.

5.1 Treewidth.

We now show that, for fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that constructs a tree-decomposition with treewidth $\leq k$, or decides that such a tree-decomposition does not exist for a given graph G. This improves on an $\mathcal{O}(n^{k+2})$ algorithm by Arnborg, Corneil and Proskurowski [2]. For $k = 1, 2, 3$ there exist linear time algorithms [4, 20]. For variable k, the problem is NP-complete [2].

Lemma 5.1 [6]
Let $G = (V, E)$ be a graph, with $W \subseteq V$ is a clique in G. Then, for any tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of G, there exists an $i \in I$ with $W \subseteq X_i$.

Our algorithm is based on the following lemma's, which are slight modifications of lemma's of Arnborg, Corneil and Proskurowski [2].

Lemma 5.2

(i) If $G = (V, E)$ has treewidth $\leq k$, and $|V| \geq k$, then there exist vertices $v_1, \ldots, v_k \in V$, with $G' = (V, E \cup \{(v_i, v_j) | 1 \leq i, j \leq k, i \neq j\}) = G \cup$ clique $(v_1, \ldots, v_k)$ has treewidth $\leq k$.

(ii) Suppose treewidth $(G) \leq k$, $v_1, \ldots, v_k$ form a clique in G. Let $V_1, \ldots, V_r$ be the sets of vertices of the connected components of $G[V - \{v_1, \ldots, v_k\}]$. Then, for all $i, 1 \leq i \leq r : G[V_i \cup \{v_1, \ldots, v_k\}]$ has treewidth $\leq k$.

(iii) Suppose treewidth $(G) \leq k, v_1, \ldots, v_k$ form a clique in G. Suppose $G[V - \{v_1, \ldots, v_k\}]$ is connected, $|V| \geq k + 2$. Then there exists a vertex $w \in V - \{v_1, \ldots, v_k\}$, such that $G' = (V, E \cup \{(v_i, w) | 1 \leq i \leq k\})$ has treewidth $\leq k$. Moreover, for each connected component V_i of $G[V - \{v_1, \ldots, v_k, w\}]$, there is at least one vertex $w_i \in \{v_1, \ldots, v_k, w\}$ with $\forall v \in V_i \ (v, w_i) \neq E$, and $G'[V_i \cup (\{v_1, \ldots, v_k, w\} - \{w_i\}]$ has treewidth $\leq k$.

We now sketch our algorithm.

1. Run the "approximate tree-decomposition " algorithm of Robertson and Seymour (see lemma 2.2). If it tells us that the treewidth of G is larger than k, then output "no", and stop. Otherwise we have a tree-decomposition of G with treewidth $\mathcal{O}(1)$.

2. Test whether G contains a minor in the obstruction set of the graphs with treewidth $\leq k$. If so, then output "no", and stop. Otherwise, we know that treewidth $(G) \leq k$, and we continue with step 3. (Step 1 and 2 basically form the recognition algorithm of Robertson and Seymour [27]).

3. Find vertices $v_1, \ldots, v_k$ as indicated in lemma 5.2(i). This can be done in linear time, using lemma 2.1, observing that

$$\exists v_1 \in V \exists v_2 \in V \cdots \exists v_k \in V : G' = (V, E \cup \{(v_i, v_j) | 1 \leq i, j \leq k, i \neq j)) hastreewidth \leq k.$$

can be written as a monadic second order graph property, by using lemma 4.1, and the characterization with forbidden minors. With k applications of lemma 2.1 we find vertices $v_1, v_2, \ldots, v_k$.

4. Determine the connected components $V_1, \ldots, V_r$ of $G[V - \{v_1, \ldots, v_k\}]$. Let $G' = G \cup \text{clique}\{v_1, \ldots, v_k\}$.

5. Now, for each $i, 1 \leq i \leq r$, find a tree-decomposition with treewidth $\leq k$ of $G'[V_i \cup \{v_1, \ldots, v_k\}]$, with a procedure, described below. Then build a tree-decomposition of G with treewidth $\leq k$ as follows:

   (a) Observe that, for each $i, 1 \leq i \leq r$, there must be a set $X_{\alpha(i)}$ in the tree-decomposition of $G'[V_i \cup \{v_i, \ldots, v_k\}]$ that contains $v_1, \ldots, v_k$, as these vertices form a clique in this graph.

   (b) Now take the disjoint union of all r tree-decompositions, add an extra set $X_o = \{v_1, \ldots, v_k\}$, and add a tree-edge from X_o to $X_{\alpha(i)}$ for all $i, 1 \leq i \leq r$. One can now check that a correct tree-decomposition of G with treewidth $\leq k$ results.

6. Next we describe a recursive procedure, that given a set $W \subseteq V$, with $G[W]$ connected, and vertices $v_1, \ldots, v_k \in V$, finds a tree-decomposition of $G' = G[W \cup \{v_1, \ldots, v_k\}] \cup \text{clique}(\{v_1, \ldots, v_k\})$, with treewidth $\leq k$. The set, associated to the root of the resulting tree contains $v_1, \ldots, v_k$.

   (a) If $|W| \leq 1$, then take the tree-decomposition $(\{X_1 = W \cup \{v_1, \ldots, v_k\}\ \}, (\{1\}, \emptyset))$.

   (b) Otherwise, find a vertex $w \in W$, with $G[W \cup \{v_1, \ldots, v_k\}] \cup$ clique $(\{v_1, \ldots, v_k, w\})$ has treewidth $\leq k$. This can be done in linear time, with the methods, exposed in sections 3 and 4. Lemma 5.2 guarantees us, that such a vertex w exists.

   (c) Determine the connected components $W_1, \ldots, W_{r'}$, of $G[W - \{w\}]$.

   (d) For each of these connected components W_i, find $W_i \in \{v_1, \ldots, v_k, w\}$ with $\forall v \in W_i (v, w_i) \notin E$. (See lemma 5.2). Now call the procedure recursively with set W_i and vertices $\{v_1, \ldots, v_k, w\} - \{w_i\}$.

   (e) So now we have tree-decompositions of all graphs $G_i = G[W_i \cup (\{v_1, \ldots, v_k, w\} - \{w_i\})] \cup \text{clique}(\{v_1, \ldots, v_k, w\} - \{w_i\})$, with treewidth $\leq k$. The root of such a tree-decomposition contains $\{v_1, \ldots, v_k, w\} - \{w_i\}$. The desired tree-decomposition of G' can be built as follows: take the disjoint union of the tree-decomposition of G_i $(1 \leq i \leq r')$. Take a new set $X_r = \{v_1, \ldots, v_k, w\}$, which is taken as the root of the new tree-decompositions. Connect X_r to each of the roots of the tree-decomposition of graphs G_i. One can check that indeed the resulting structure is a tree-decomposition of G' with treewidth $\leq k$.

This completes the description of the algorithm. The time needed for steps 1,2,3,4 and 5 is bounded by $\mathcal{O}(n^2)$. Each call of the procedure in step 6 costs $\mathcal{O}(n)$ time, and this procedure is called $\mathcal{O}(n)$ times. Thus, the total time of our algorithm is $\mathcal{O}(n^2)$.

Theorem 5.3
For each constant k, there exists an $\mathcal{O}(n^2)$ algorithm, that for a given graph $G = (V, E)$ either decides that the treewidth of G is larger than k, or finds a tree-decomposition of G with treewidth $\leq k$.

(In [8] an (easier) $\mathcal{O}(n^3)$ algorithm is given.)

5.2 Search number and vertex search number.

In this section we consider the search number and vertex search number of a graph. A search strategy of a graph is a sequence of the following types of moves:

1. Place a searcher on a vertex.

2. Delete a searcher from a vertex.

3. Move a searcher over an edge.

All edges are initially *contaminated.* An edge (v, w) can become *cleared* by moving a searcher from v to w, while there is a second searcher on v, or all other edges, adjacent to v are already cleared. An edge can become *recontaminated*, when a move results in a path without searchers from a contaminated edge to the edge. The search number of G is the minimum number of searchers needed to clear all edges. It has been shown by LaPaugh [17] that for every graph G, there exists a search sequence, that uses the optimal number of searchers, and does not allow recontamination. Such a search sequence is called *progressive.* If we let vertices instead of edges be cleared or contaminated, then the vertex search number is the minimum number of searchers, needed to clear all vertices with a progressive search sequence. Determining the minimum number of searchers needed is NP-complete [21]. Note that for a progressive search strategy that clears all vertices, we may assume that never two searchers occupy the same vertex, and that once a searcher has left a vertex, then no searcher will visit that vertex again. Note that, for fixed k, the classes of graphs with search number of vertex search number $\leq k$ are closed under taking of minors. Also, if the (vertex) search number of G is k, then treewidth $(G) \leq k$ (see e.g. [9]).

Definition.
$BW(n, k) = (\{v_1, \ldots, v_n\}, \{(v_i, v_j) | 1 \leq i, j \leq n, |i - j| \leq k\})$.

($BW(n, k)$ is the maximal graph on n vertices with bandwidth k.)

Lemma 5.4

(i) If there exists a progressive search strategy that clears all vertices of $G = (V, E)$ with k searchers, then there exists one with the first k moves the placing of a searcher ($|V| \geq k$).

(ii) There exists a progressive search strategy that clears all vertices of G, and that starts with placing a searcher on vertices $w_1, \ldots, w_k$, if and only if the graph G', obtained by taking the disjoint union of G and $BW(2k+1, k)$ and then identifying vertices v_i and w_i for $1 \leq i \leq k$, has vertex search number $\leq k$.

Proof.
(i) One can obtain the desired search strategy by first executing the first k moves of the type "place a searcher on a vertex", and then executing all other moves in sequence.
(ii) $\Rightarrow$ Use the following search strategy: place searchers on vertices $v_{k+2}, v_{k+3}, \ldots, v_{2k+1}$. Then move a searcher from v_i to v_{i-k}, for $i = 2k+1, 2k, 2k-1, \ldots, k+1$. Now we have searchers on $w_1 = v_1, \ldots, w_k = v_k$. Continue with the given search strategy that clears all vertices of G and starts with searchers on $w_1, \ldots, w_k$.
$\Leftarrow$ Consider a progressive search strategy that clears all vertices of G' with k searchers. Consider the first move that removes a searcher from a vertex $v_i \in \{v_{k+1}, \ldots, v_{2k+1}\}$ or moves a searcher away from a vertex $v_i \in \{v_{k+1}, \ldots, v_{2k+1}\}$. It follows that after this move, all vertices in $\{v_{k+1}, \ldots, v_{2k+1}\} - \{v_i\}$ must contain a searcher. Also, vertices $v_{i-1}, \ldots, v_{i-k}$ must be cleared or contain a searcher. It follows that either all vertices in G are cleared or uncleared. Suppose the latter. It follows that $i = 2k+1$, and that the next $k+1$ moves are: move a searcher from v_j to v_{j-k}, for $j = 2k+1, \ldots, k+1$. We then have searchers on $v_1 = w_1, \ldots, v_k = w_k$. The remaining search sequence is a progressive search sequence that clears all vertices of G, starting with searchers on $v_1, \ldots, v_k$. In the case that all vertices in G are cleared after moving the searcher from v_i, we use the same argument for the search strategy, that is obtained by "reversing" the original strategy. □

Lemma 5.4 gives us a method to find the first k vertices where a searcher is placed: we must find vertices $v_1, \ldots, v_k$, such that the graph $(V \cup \{w_{k+1}, \ldots, w_{2k+1}\}, E \cup \{(v_i, v_j) | 1 \leq i, j \leq k, i \neq j\} \cup \{(v_i, w_j) | 1 \leq i \leq k < j \leq i + k\} \cup \{(w_i, w_j) | k \leq i, j \leq 2k+1, 0 < |i-j| \leq k\})$ has vertex search number k. This can be done in linear time, by combining the techniques of section 3 and 4. (Add $w_{k+1}, \ldots, w_{2k+1}$ to G, and all edges between them. Construct a constant width tree-decomposition of the resulting graph G'. Write: "$\exists v_1 \ldots \exists v_k$: if we add edges between $v_i, v_j, 1 \leq i \leq j \leq k$, and v_i, w_j with $j - i \leq k$, then the resulting graph has vertex search number $\leq k$" as a monadic second order graph property, using the characterization with forbidden minors, and lemma 4.1.).

In order to find the next move, we distinguish between two cases.

Case 1: There exists a vertex v_i, containing a searcher, that is not adjacent to a vertex that does not contain a searcher. Then one may assume that the next move in our search strategy is the removal of the searcher from v_i. Now remove v_i from G, and find, recursively a progressive search strategy that clears all vertices of $G - \{v\}$ with k searchers, that starts with placing a vertex on each of the vertices $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k$. (This can be done, similar as how we found the first k moves. Only let $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k$ be free variables in the monadic second order graph property that is tested.) The resulting search sequence can easily be extended to the desired search sequence.

Case 2: Such a vertex v_i does not exists. Then necessarily the next move must be the moving of a searcher from a vertex v_i to an adjacent uncleared vertex. Note that for each searcher, there is at most one vertex where it can move to, otherwise the vertex it leaves will become recontaminated. So we have to consider at most k possible moves. For each possible move from v_i to w, test whether there exists a progressive search strategy that clears the vertices of $G - \{v_i\}$ with k searchers, and that starts with placing a searcher on $v_1, \ldots, v_{i-1}, v_{i+1}, v_k$, and w. This can be tested, similar as above, in linear time. Do this for each of the k possible moves, until a good move is found. Then find recursively a progressive search strategy, that clears the vertices of G with k searchers, that starts with placing a searcher on $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k$ and w.

As in total $\mathcal{O}(n)$ moves are made, the total time of the algorithm is $\mathcal{O}(n^2)$.

Theorem 5.5
For each constant k, there exists an $\mathcal{O}(n^2)$ algorithm, that for a given graph $G = (V, E)$, either decides that the vertex search number of G is larger than k, or finds a progressive search sequence that clears the vertices of G with k searchers.

We can use this result to obtain the following theorem as an easy corollary.

Theorem 5.6
For each constant k, there exists an $\mathcal{O}(n^2)$ algorithm, that for a given graph $G = (v, E)$ either decides that the search number of G is larger than k, or finds a progressive search strategy that clears all edges of G with k searchers.

Proof.
Let G' be the graph, obtained by subdividing each edge in G once, i.e. $G' = (V \cup E, \{(v, e) | v \in V, e \in E, \exists w \in V : (v, w) = e\})$. The vertex search number of G' equals the search number of G, and the corresponding search strategies can be easily transformed into each other. □

5.3 Other problems.

Several other problems can be dealt with in the same manner. The techniques are similar to those used for the vertex search number problem, but the details are different.

We give the definitions of the problems, that are considered. A linear ordering of a graph $G = (V, E)$ is a bijection $V \to \{1, 2, \ldots, |V|\}$. The cutwidth of a linear ordering f is $\max_{1 \leq i < n} |\{(v, w) \in E | f(v) \leq i < f(w)\}|$. The cutwidth of G is the minimum cutwidth over all linear orderings of G. The vertex separation of a linear ordering f is $\max_{1 \leq i < n} |\{v \in V | f(v) \leq i \wedge \exists w \in V : (v, w) \in E \cap f(w) > i\}|$. The vertex separation number of G is the minimum vertex separation over all linear orderings of G. The modified cutwidth of a linear ordering f is $\max_{1 \leq i < n} |\{(v, w) \in E | f(v) < i < f(w)\}|$. The modified cutwidth of G is the minimum modified cutwidth over all linear orderings of G. A path-decomposition of G is a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of G, where T is a path, i.e. T is a tree with every node degree 1 or 2. The pathwidth of path-decomposition $(\{X_i | i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The pathwidth of G is the minimum pathwidth over all path-decompositions of G.

Determining the cutwidth, modified cutwidth, vertex separation number or pathwidth of a graph is NP-complete (see e.g. [2, 22]). For fixed k, Fellows and Langston obtained $\mathcal{O}(n^3)$ and $\mathcal{O}(n^3 \log n)$ algorithms [12, 16].

Theorem 5.7

(i) For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that for a given graph $G = (V, E)$ either decides that the cutwidth of G is larger than k, or finds a linear ordering of G with cutwidth $\leq k$.

(ii) For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that for a given graph $G = (V, E)$ either decides that the pathwidth of G is larger than k, or finds a path-decomposition of G with pathwidth $\leq k$.

(iii) For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that for a given graph $G = (V, E)$ either decides that the vertex separation number of G is larger than k, or finds a linear ordering of G with vertex separation $\leq k$.

(iv) For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm either decides that for a given graph $G = (V, E)$, the modified cutwidth of G is larger than k, or finds a linear ordering of G with modified cutwidth $\leq k$.

Proof.

(i) Use the self-reduction of Brown, Fellows and Langston [10], and apply the techniques of section 3 and 4. Parallel edges can be avoided by putting a vertex on the middle of these edges.

(ii), (iii), (iv) Omitted. □

Corollary 5.8
For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm, that for a given Boolean matrix M, either finds a column permutation of M such that if in each row every 0 lying between the row's leftmost and rightmost 1 is changed to a *, then no column contains more than k 1's and *s, or decides that such a column permutation does not exist.

Proof.
There is a linear transformation from this problem to the problem of finding path-decompositions with pathwidth $\leq k-1$ [14, 16]. □

Corollary 5.9
For every fixed k, there exists an $\mathcal{O}(n^2)$ algorithm that, for a given directed acyclic graph $G = (V, E)$, either gives a pebbling strategy for the progressive black-white pebbling game on G, that uses k pebbles (see [19] for a definition) or decides that such pebbling strategy does not exist.

Proof.
There is a linear transformation from this problem to the vertex separation problem and vice versa [19]. □

6 Final remarks.

As discussed in section 1, the algorithms are non-constructive in the sense that one knows that an algorithm exists, but we do not have a concrete algorithm of which we know that it is correct. In order to overcome this problem, Fellows and Langston [12] designed a technique that allows us to actually construct algorithms without knowing the exact obstruction set. The technique does not work always, but only if we have an (efficient) oracle algorithm, producing "solutions", an (efficient) algorithm that checks whether a "candidate-solution" is a correct solution, and an arbitrary (other) algorithm, that decides on membership in the considered class of graphs. The resulting algorithm has the following form. Let T be a minor or immersion closed class of graphs.

1. Let S be a subset of the obstruction set of T.
2. Check whether the input graph G has a graph in S as a minor. If so, output "no" ($G \neq T$), and stop.
3. Use the oracle algorithm to produce a 'solution', using S as an obstruction set.

4. Check the produced solution. If it is a correct solution, output it ($G \in T$), and stop.

5. (S is not the complete obstruction set of T). Enumerate all graphs, until a graph $G, G \notin S, G \notin T$, all minors of G are in T, are found. (G is a new element of the obstruction set). Put G in S. Go to step 2.

We can apply this technique to each of the problems considered in section 5. Observe that the method of Arnborg, Lagergren and Seese [3] that obtains a linear time algorithm on bounded treewidth graphs, given an extended monadic second order graph property, is in fact an automatic procedure. As our obstruction sets may grow, our monadic second order graph properties can vary during this algorithm, so we need to implement this procedure in order to combine the techniques, and obtain fully constructive $\mathcal{O}(n^2)$ algorithms for the problems considered in section 5.

References

[1] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - A survey. *BIT*, 25:2–23, 1985.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[3] 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.

[4] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *Siam J. Alg. Disc. Meth.*, 7:305–314, 1986.

[5] 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.

[6] H. L. Bodlaender. *Dynamic programming algorithms on graphs with bounded tree-width.* Tech. Rep., Lab. for Comp. Science, M.I.T., 1987. Ext. abstract in proceedings ICALP 88.

[7] H. L. Bodlaender. *NC-algorithms for graphs with small treewidth.* Technical Report RUU-CS-88-4, Dept. of Comp. Science, Univ. of Utrecht, Utrecht, 1988. To appear in: Proc. Workshop on Graph-Theoretical Concepts in Computer Science, 1988.

[8] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. In *Proc. 1st Scandinavian Workshop on Algorithm Theory*, pages 223–232, 1988.

[9] H. L. Bodlaender. Some classes of graphs with bounded treewidth. *Bulletin of the EATCS*, 1988. To appear.

[10] D. Brown, M. Fellows, and M. Langston. Nonconstructive polynomial-time decidability and self-reducibility. In *Princeton Forum on Algorithms and Complexity*, 1987.

[11] 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.

[12] M. R. Fellows and M. A. Langston. Fast self-reduction algorithms for combinatorial problems of VLSI design. In *Proc. 3rd Aegean Workshop on Computing*, 1988.

[13] M. R. Fellows and M. A. Langston. Layout permutation problems and well-partially-ordered sets. In *5th MIT Conf. on Advanced Research in VLSI*, pages 315–327, 1988.

[14] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Inform. Proc. Letters*, 26:157–162, 1987.

[15] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. 1987. To appear in JACM.

[16] M. R. Fellows and M. A. Langston. On seach, decision and the efficiency of polynomial-time algorithms. 1988. Extended abstract.

[17] A. S. LaPaugh. *Recontamination Does Not Help to Search a Graph.* Technical Report, Comp. Sci. Dept, Princeton Univ., New Yersey, 1982.

[18] C. Lautemann. Efficient algorithms on context-free graph languages. In *Proc. 15'th ICALP*, pages 362–378, Springer Verlag, Lect. Notes in Comp. Sc. 317, 1988.

[19] T. Lengauer. Black-white pebbles and graph separation. *Acta Inf.*, 16:465–475, 1981.

[20] J. Matousěk and R. Thomas. Algorithms finding tree-decompositions of graphs. 1988. Unpublished paper.

[21] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35:18–44, 1988.

[22] B. Monien and I. H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theor. Comp. Sc.*, 58:209–229, 1988.

[23] N. Robertson and P. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. of Algorithms*, 7:309–322, 1986.

[24] N. Robertson and P. Seymour. Graph minors. III. Planar tree-width. *J. Combin. Theory Series B*, 36:49–64, 1984.

[25] N. Robertson and P. Seymour. Graph minors. IV. Tree-width and well-quadi-ordering. Manuscript.

[26] N. Robertson and P. Seymour. Graph minors. X. Obstructions to tree-decompositions. 1986. Manuscript.

[27] N. Robertson and P. Seymour. Graph minors. XIII. The disjoint paths problem. 1986. Manuscript.

[28] N. Robertson and P. Seymour. Graph minors. XVI. Wagner's conjecture. To appear.

[29] P. Scheffler. *Linear-time algorithms for NP-complete problems restricted to partial k-trees.* Report R-MATH-03/87, Karl-Weierstrass-Institut Für Mathematik, Berlin, GDR, 1987.