On reduction algorithms for graphs with small treewidth*

Hans L. Bodlaender

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

Abstract. Some new ideas are presented on graph reduction applied to graphs with bounded treewidth. It is shown that the method can not only be applied to solve decision problems, but also some optimization problems and construction variants of several of these problems on graphs with some constant upper bound on the treewidth.
Also, the exisitence is shown of finite, safe, complete, and terminating sets of reduction rules, such that on any graph G with treewidth at most some constant k, $\Omega(n)$ applications of reduction rules can be applied simultaneously. This result is used to obtain a class of randomized parallel algorithms that solve many problems on graphs with bounded treewidth and bounded degree in $O(\log n)$ expected time with $O(n)$ processors on a EREW PRAM. Among others, such an algorithm is obtained to recognize the class of graphs with treewidth at most k and maximum vertex degree at most d, for constant k and d.

1 Introduction

In this paper, new ideas and results are presented on graph reduction, applied to graphs with bounded treewidth.

We consider reduction rules, where a connected subgraph of a graph G is to be replaced by another smaller subgraph (under some additional rules, see Section 2 for the precise definitions.) Arnborg et al [2] showed that for each property P, which is 'finite index' and each constant k, there exists a finite, complete, safe, and terminating set of reduction rules for graphs with treewidth at most k: a graph G is reduced by a series of applications of reduction rules from the set to a graph from some finite set of 'small' graphs, if and only if $P(G)$ holds and the treewidth from G is at most k. This result is used to show the existence of linear time algorithms, that decide whether property P holds for a given graph G with bounded treewidth, without the need of using a tree-decomposition of G. It should be noted that the algorithm uses more than linear memory. (The set of finite index properties includes many interesting properties, including all properties expressible in monadic second order logic.)

In this paper, we extend these results from [2] in three ways:

– We show that a variant of the method can be used to solve several optimization problems.

* This work was partially supported by the ESPRIT Basic Research Actions of the EC under contract 7141 (project ALCOM II).

- We discuss a method to solve in many cases also the construction variants of the problem.
- We show the existence of finite, complete, safe, and terminating sets of reduction rules such that on any graph with treewidth at most k in the class to be recognized, $\Omega(n)$ applications of a reduction rule from the set can be applied simultaneously.

The latter result leads to a class of randomized parallel algorithms, that decide on finite index properties, or solve the optimization problems mentioned above, in $O(\log n)$ expected time with $O(n)$ processors on a CRCW PRAM, on graphs with bounded treewidth and bounded degree. These include an algorithm with this time and processor bounds that recognizes the class of graphs with treewidth at most k and degree at most d. This latter result should be compared with the following existing results:

- an algorithm that uses $O(\log n)$ time, but $O(n^{3k+1})$ processors [4],
- an algorithm that uses $O(\log^2 n)$ time and $O(n/\log n)$ processors (which can be obtained by a parallel implementation of the algorithm in [9]),
- a sequential algorithm, that uses $O(n)$ time [6].

Each of the three algorithms mentioned above does not require a bound on the degree of the input graph. So, although the result presented in this paper has its clear limitations, it is the first parallel algorithm for the problem that uses $O(\log n)$ expected time and loses only a (poly-)logarithmic factor in its processor-time product, compared with the time of the best sequential algorithm.

See [5, 7] for more backgrounds on graph reduction, and graphs of bounded treewidth.

2 Preliminaries

In this paper, the graphs we consider are undirected, do not contain self-loops or multiple edges. (Similar results can be derived for directed graphs. For simplicity, we concentrate on undirected graphs.)

With $d(G)$, we denote the maximum degree over all vertices of G.

The notion of treewidth was introduced by Robertson and Seymour [10].

Definition. A tree-decomposition of a graph $G = (V, E)$ 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, one for each node of T, and T a tree such that

- $\bigcup_{i \in I} X_i = V$.
- for all edges $(v, w) \in E$, there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$.
- for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

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

Definition. A *terminal graph* is a triple (V, E, X) with (V, E) an undirected graph, and $X \subseteq V$ is an ordered subset of the vertices, called the set of *terminals*. Vertices in $V - X$ are called *inner vertices*. Terminal graph (V, E, X) is called a k-terminal graph, if $|X| = k$. A terminal graph (V, E, X) is said to be *open*, if there are no edges between terminals ($X \times X \cap E = \emptyset$).

The usual undirected graphs (i.e., without terminals) will be simply called *graph*.

Definition. The operation $\oplus$ maps two terminal graphs with the same number of terminals to a graph, by taking the disjoint union of the two graphs and then identifying the corresponding terminals, i.e., for $i = 1 \cdots k$, the ith terminal of the first terminal graph is identified with the ith terminal of the second graph (k the number of terminals).[2]

For an example, see figure 1.

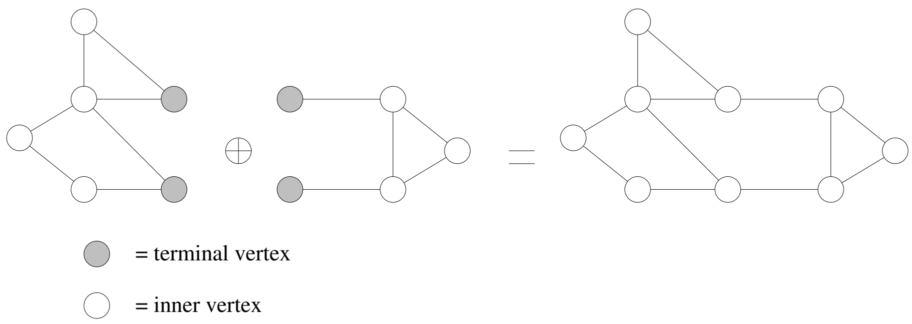

Fig. 1. Example of $\oplus$ operation

Two terminal graphs $(V_0, E_0, < x_1, \cdots, x_k >)$ and $(V_1, E_1, < y_1, \cdots, y_l >)$ are said to be *isomorphic*, if $k = l$ and there exists a function $f : V_0 \rightarrow V_1$ with for all $v, w \in V_0 : (v, w) \in E_0 \Leftrightarrow (f(v), f(w)) \in E_1$ and for all i, $1 \leq i \leq k : f(x_i) = y_i$. (The main difference between the usual definition of graph isomorphism is that we require that the corresponding terminals are mapped to each other.)

A *reduction rule* A is an ordered pair of two open terminal graphs with the same number of terminals. It is denoted as $G_0 \stackrel{A}{\rightarrow} G_1$, or if A is clear from the context as $G_0 \rightarrow G_1$. An application of rule $G_0 \rightarrow G_1$ is the operation, that takes a graph H of the form $H_0 \oplus H_2$, with H_0 isomorphic to G_0, and replaces it by the graph $H_1 \oplus H_2$, with H_1 isomorphic to G_1. We write $H \stackrel{A}{\rightarrow} H_1 \oplus H_2$. An example is given in figure 2.

For a set of reduction rules $\mathcal{A}$, we write $H \stackrel{\mathcal{A}}{\rightarrow} H'$, if there exists an $A \in \mathcal{A}$ with $H \stackrel{A}{\rightarrow} H'$.

Let P be a property of graphs. We say that the set of reduction rules $\mathcal{A}$ is *safe* for P, if whenever $H \stackrel{\mathcal{A}}{\rightarrow} H'$, then $P(H) \Leftrightarrow P(H')$. The set is said to be *complete*

[2] The $\oplus$ operation could create a graph with multiple edges between a pair of vertices. However, by using the $\oplus$ operation only on pairs of terminal graphs of which at least one is open, we will make sure, that this never happens.

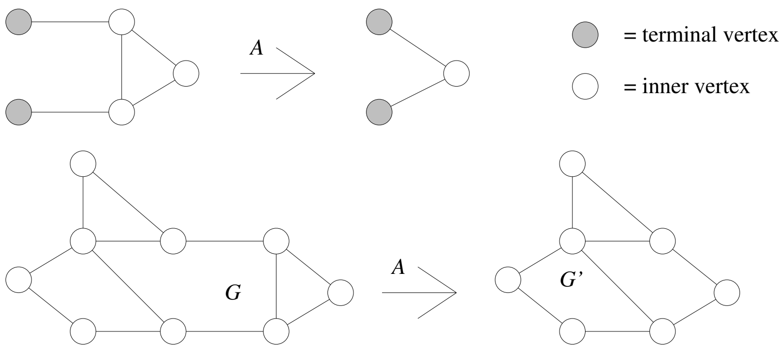

Fig. 2. Applying rule A to G yields G'

for P, if the set of graphs $\{H \mid P(H) \wedge \neg\exists H' : H \stackrel{\mathcal{A}}{\to} H'\}$ is finite. It is said to be *terminating*, if there does not exist an infinite sequence $H_1 \stackrel{\mathcal{A}}{\to} H_2 \stackrel{\mathcal{A}}{\to} H_3 \stackrel{\mathcal{A}}{\to} \cdots$. It is said to be *decreasing*, if whenever $H \stackrel{\mathcal{A}}{\to} H'$, then H' contains fewer vertices than H. Clearly, a decreasing set of rules is terminating.

A set of reduction rules $\mathcal{A}$, that is finite, safe, complete, and terminating for a property P corresponds to an algorithm that decides whether property P holds on a given graph: repeat applying rules from $\mathcal{A}$ starting with the input graph, until no rule from $\mathcal{A}$ can be applied anymore. If the resulting graph belongs to the finite set $\{H \mid P(H) \wedge \neg\exists H' : H \stackrel{\mathcal{A}}{\to} H'\}$, then P holds on the input graph, otherwise it does not. In [2] it is shown how when the set is decreasing, this algorithm can be implemented, such that it takes linear time and polynomial space.

For any property P of graphs, we define the equivalence relation $\equiv_{P,l}$ on l-terminal graphs, as follows:

$$H_1 \equiv_{P,l} H_2 \Leftrightarrow (\forall\ l\text{-terminal graphs } K : P(G \oplus K) \Leftrightarrow P(H \oplus K))$$

We say that a property P is of *finite index*, if for all l: $\equiv_{P,l}$ has a finitely many equivalence classes. It appears that many important graph properties are finite index. For instance, all properties that can be formulated in monadic second order logic are finite index. These include Hamiltonicity, k-colorability (for fixed k), and many others. (See e.g. [3].)

For a property P, the property $P_{tw,K}$ is defined as $P_{tw,K}(G) = P(G) \wedge tw(G) \leq K$.

Lemma 1. *If P is finite index, then $P_{tw,K}$ is finite index.*

(A similar lemma holds, if we pose an additional constant upper bound on the maximum degree of vertices in the graph.)

Finite index corresponds to 'finite state': there exists a linear time algorithm that decides the property on graphs, given with a tree-decomposition of bounded treewidth. Moreover, this algorithm is of a special, well described structure. See e.g. [1].

Safe reduction rules are implied by the equivalence relation $\equiv_{P,l}$: if for l-terminal graphs G_1, G_2 it holds that $G_1 \equiv_{P,l} G_2$, then it directly follows from the definitions that the reduction rule $G_1 \rightarrow G_2$ is safe.

Reduction rules for optimization problems We now extend the idea of graph reduction to optimization problems. Let R be a function, mapping the set of graphs to $\mathbf{Z} \cup \{$ **false**$\}$. Typically, R will be an optimization problem, like independent set, vertex cover, etc. The value **false** is used to denote that a certain condition does not hold, especially it is used to deal with graphs that have treewidth more than the fixed upper bound K. Denote $\mathcal{Z} = \mathbf{Z} \cup \{\, \mathbf{false}\}$. Define addition on $\mathcal{Z}$ as follows: if $i, j \in \mathbf{Z}$, then we take for $i + j$ the usual sum, and for all $i \in \mathcal{Z} : i+$ **false** $=$ **false**$+i =$ **false**.

We say that R is *finite integer index*, if for each constant l, there exists a finite set S_l and a function κ_l that maps each l-terminal graph to a pair $(s, i) \in S_l \times \mathcal{Z}$, such that for all l-terminal graphs G_1, G_2, and for all $s \in S_l$, $i, j \in \mathcal{Z}$: if $\kappa_l(G_1) = (s, i_1)$ and $\kappa_l(G_2) = (s, i_2)$, then for all l-terminal graphs H,

$$R(G_1 \oplus H) - i_1 = R(G_2 \oplus H) - i_2$$

As a shorthand notation, we write $G_1 \equiv^{+i}_{R,k} G_2$, if there exist $s \in S_k$, $i_0 \in \mathcal{Z}$ with $\kappa_k(G_1) = (s, i_0 + i)$ and $\kappa_k(G_2) = (s, i_0)$. We call $\kappa_k(G)$ the integer index of k-terminal graph G.

The idea is now to maintain with the (possibly reduced) input graph an integer variable (futher denoted as: 'the counter'). We now take reduction-counter rules $G_1 \rightarrow^{+i} G_2$, that not only carry out the rule $G_1 \rightarrow G_2$ as described above, but also add i to the counter. (Formally, a reduction-counter rule is a pair, consisting of an integer, and an ordered pair of open terminal graphs with the same number of terminals.) Notations as for reduction rules are used here in the same way.

We say a set of reduction-counter rules B is *safe*, if for all rules $B \in \mathcal{B}$, for all graphs H_1, H_2, if $H_1 \overset{B}{\rightarrow}^{+i} H_2$, then $R(G_1) = R(G_2) + i$. The set is said to be *complete*, if the set $\{H \mid R(H) \neq \mathbf{false} \wedge \neg\exists H', i \in \mathbf{Z}\text{: } H \overset{B}{\rightarrow}^{+i} H'\}$ is finite. The definitions of *terminating* and *decreasing* are similar to the finite index case.

If for a finite integer index optimization function R, we have for two k-terminal graphs, $G_1 \equiv^{+i}_{R,k} G_2$, then the rule $G_1 \rightarrow^{+i} G_2$ is safe. The rule corresponds to changing the k-terminal subgraph isomorphic to G_1 to a subgraph, isomorphic to G_2, and simultaneously adding i to the counter.

It is important to note that the sum of $R(G)$ and the counter does not change under safe reduction-counter rules. Thus, when G has been rewritten to a small graph G', one can determine $R(G)$ by calculating $R(G')$ and adding the counter to this number.

For a finite integer index function R, the funtion $R_{tw,K}$ is defined as

$$R_{tw,K} = \begin{cases} \mathbf{false}, & \text{if } tw(G) > K \\ R(G) & \text{otherwise} \end{cases}$$

Lemma 2. *If R is finite integer index, then $R_{tw,K}$ is finite integer index.*

Some useful lemmas on graphs with bounded treewidth Below, we give two lemmas on the existence of subgraphs of a certain size and type in graphs with bounded treewidth. These lemmas will be used later in the paper.

Lemma 3. *Let k, r be positive integers. If $G = (V, E)$ is a graph with n vertices and treewidth $\leq k$, $n \geq r+1$, then G can be written as $G_1 \oplus G_2$, with G_1 and G_2 terminal graphs with at most $k+1$ terminals, and G_1 has at least $r+1$ and at most $2r+k$ vertices.*

The following lemma basically sais that in a graph with small treewidth, one can find 'many' terminal graphs as subgraph, that have size and number of terminals between certain bounds, and do not share inner vertices.

Lemma 4. *Let k, r be positive integers. If $G = (V, E)$ is a graph with n vertices and treewidth $\leq k$, $n \geq r+1$, then there exists a collection $\mathcal{C}$ of at least $\frac{|V|}{36(k+1)r}$ pairs of sets of vertices (W, Z), such that*

1. *If (W_1, Z_1), $(W_2, Z_2) \in \mathcal{C}$ and $(W_1, Z_1) \neq (W_2, Z_2)$, then $W_1 \cap W_2 = \emptyset$.*
2. *If $(W, Z) \in \mathcal{C}$, then $r \leq |W \cup Z| \leq 18r(k+1)$.*
3. *If $(W, Z) \in \mathcal{C}$, then $|Z| \leq 6k+6$.*
4. *If $(W, Z) \in \mathcal{C}$, then for all $(v, w) \in E : v \in W \Rightarrow w \in W \cup Z$.*
5. *If $(W, Z) \in \mathcal{C}$, then $G[W \cup Z]$ is a connected subgraph of G.*

3 Finite, safe, complete, and terminating sets of reduction rules

In this section, we show that for each finite index property P there exists a finite, safe, complete and decreasing set of reduction rules for P, and for each finite integer index function R, there exists a finite, safe, complete, and decreasing set of reduction-counter rules for R. The first of these results has been shown already by Arnborg et al [2]; the proof given below may be somewhat easier to follow.

Theorem 5. *Let k be a constant.*

(i) [2] If P is finite index, then there exists a finite, safe, complete and decreasing set of reduction rules $\mathcal{A}$ for $P_{tw,k}$. Moreover, all terminal graphs in a left-hand-side of a rule in $\mathcal{A}$ are connected.

(ii) If R is finite integer index, then there exists a finite, safe, complete and decreasing set of reduction-counter rules $\mathcal{B}$ for $P_{tw,k}$. Moreover, all terminal graphs in a left-hand-side of a rule in $\mathcal{B}$ are connected.

Proof. (i) For every $l \leq k+1$, and every equivalence class Q of $\equiv_{P_{tw,k},l}$ that contains at least one open connected l-terminal graph, choose a 'representing' open connected l-terminal graph $G_Q \in Q$. Let r be the maximum number of vertices of a graph G_Q over all $l \leq k+1$ and all these equivalence classes Q. Now, for all $l \leq k+1$, for all open connected l-terminal graphs G with at least $r+1$ and at most $2r+k$ vertices, add the reduction rule $G \to G_Q$ to a set of reduction rules $\mathcal{A}$.

Note that, as each right-hand-side of a rule in $\mathcal{A}$ is open, applying a rule in $\mathcal{A}$ can never give multiple edges between a pair of vertices. It is also easy to see that $\mathcal{A}$ is finite: there are finitely many l-terminal graphs with at most $2r+k$ vertices. Safeness of the resulting set $\mathcal{A}$ follows directly from the fact that each left- and right-hand-side of a rule in $\mathcal{A}$ belong to the same equivalence class of a relation $\equiv_{P_{tw,k},l}$. As, by lemma 3, each graph with treewidth at most k and at least $r+1$ vertices, has an applicable rule from the set $\mathcal{A}$, completeness follows directly: the set $\{H \mid P(H) \ \wedge \ \neg\exists H' : H \overset{\mathcal{A}}{\to} H'\}$ contains only vertices with at most r vertices. It is obvious that $\mathcal{A}$ is decreasing.

(ii) For every $l \leq k+1$, consider set S_l and function κ_l as in the definition of finite integer index, for the problem $R_{tw,k}$. For each $s \in S_l$ such that there exists an open connected l-terminal graph G_s with $\kappa_l(G_s) = (s,i)$ with $i \in \mathbf{Z}$, choose such a 'representing' open connected l-terminal graph G_s. Again, let r be the maximum number of vertices over all representing graphs G_s. For all $l \leq k+1$, for all open connected l-terminal graphs G with at least $r+1$ and at most $2r+k$ vertices with $\kappa_l(G) = (s,i)$ for some $s \in S_l$, $i \in \mathbf{Z}$, add the reduction-counter rule $G \to^{+i-i'} G_Q$ to a set of reduction-counter rules $\mathcal{B}$. As above, it follows that the resulting set is finite, safe, and decreasing. It is complete, because when for a graph H it holds that $\neg\exists H', i \in \mathbf{Z}$: $H \overset{\mathcal{B}}{\to}^{+i} H'$, then either H has at most r vertices, or H can be written as $H_1 \oplus H_2$, with $\kappa_l(H_1) = (s, \mathbf{false})$ for some $l \leq k+1$, $s \in S_l$. In the latter case, $R(H) = \mathbf{false}$. $\square$

Note that the sets A and B correspond to linear time algorithms that solve P or R on graphs with treewidth at most k [2]. It should be remarked that these algorithms need more than linear memory, and that linear time and linear space algorithms can be obtained by first finding a tree-decomposition of treewidth at most k [6].

4 Finite integer index problems

In this section we give some examples of problems that are finite integer index, and of some problems that can be shown not to be finite integer index.

Theorem 6. *The Maximum Independent Set problem is finite integer index.*

Proof. Recall that the size of a maximum independent set in a graph G is denoted by $\alpha(G)$. The integer index of a k-terminal graph $G = (V, E, X)$ is the pair

$$(f : \mathcal{P}(X) \to \{0, 1, \ldots, |X|\}, \alpha(G[V-X]))$$

with for all $Y \subseteq X$:

$$f(Y) = \alpha(G[V - X \cup Y]) - \alpha(G[V-X])$$

We first have to prove correctness of this definition. Clearly, for all $Y \subseteq X$, $f(Y) \geq 0$. Next note, that if Z is an independent set in $G[V - X \cup Y]$, then $Z - Y$ is an independent set in $G[V-X]$, hence $\alpha(G[V - X \cup Y]) - \alpha(G[V-X]) \leq |Y| \leq |X|$.

Consider k-terminal graphs $G_1 = (V_1, E_1, X)$ and $G_2 = (V_2, E_2, X)$ with integer index (f, r_1) and (f, r_2) respectively. We must show that for all k-terminal graphs

$H = (W, F, X)$, $\alpha(G_1 \oplus H) - r_1 = \alpha(G_2 \oplus H) - r_2$. Suppose Z is a maximum independent set in $G_1 \oplus H$. Note that there exists an independent set Z' of size $r_1 + f(Z \cap X_1)$ with $Z' \cap (X - Z) = \emptyset$. The size of $Z \cap V$ must be precisely $r_1 + f(Z \cap X)$: it cannot be more, by definition of r_1 and f, and it cannot be less, because then $Z - (V \cap Z) \cup Z'$ would be an independent set in $G \oplus H$ of size, larger than the size of Z.

There also exists an independent set Z'' of size $r_2 + f(Z \cap X)$ in G_2 with $Z'' \cap (X - Z) = \emptyset$. Now $Z - (Z \cap V) \cup Z''$ is an independent set in $G_2 \oplus H$ of size $|Z| + r_2 - r_1$. Hence $\alpha(G_1 \oplus H) - r_1 \leq \alpha(G_2 \oplus H) - r_2$. In the same way one can prove that $\alpha(G_1 \oplus H) - r_1 \geq \alpha(G_2 \oplus H) - r_2$. □

Without proof, we mention that each of the following problems is finite integer index: Partition into Cliques, Vertex Cover, Dominating Set, Covering by Cliques, Hamiltonian Completion Number. The Maximum Cut is finite integer index for graphs with bounded degree. It is also possible to show for some problems that they are not finite integer index. Notable examples are Longest Path, Longest Cycle, Steiner Tree.

5 Simultaneous applications of reductions and parallel algorithms

Two applications of reduction rules (or reduction-counter rules) on the same graph are said to be *concurrent*, if the subgraphs to be rewritten do not share any vertex that is non-terminal in at least one of the subgraphs. A collection of applications of reduction(-counter) rules is said to be concurrent, if the applications are pairwise concurrent.

The idea behind concurrent applications of rules is that in a parallel algorithm, all reduction steps from a concurrent set can be carried out simultaneously. This is very useful in order to obtain fast parallel algorithms, based on reduction. The following theorem shown that there exist sets, which always allow a linear number of concurrent reductions.

Theorem 7. *Let K be a constant. If P is finite index (R is finite integer index), then there exists a finite, safe, complete, and decreasing set of reduction rules (reduction-counter rules) $\mathcal{A}$ for $P_{tw,K}$ ($R_{tw,K}$), such that*

- *All terminal graphs in a left-hand-side of a rule in $\mathcal{A}$ are connected.*
- *There exist constants $c \in \mathbf{R}^+$, $c' \in \mathbf{N}$ such that for all graphs $G = (V, E)$ with treewidth at most K, there exists a concurrent set of $c \cdot |V|$ applications of rules from $\mathcal{A}$, or $|V| \leq c'$.*

Proof. Let P be finite index. (The proof for finite integer index is very similar.) For every equivalence class Q of $\equiv_{P_{tw,k},l}$ ($l \leq 6k + 6$), choose a representing l-terminal graph $G_Q \in Q$. Let r be the maximum size of a representing G_Q over all these equivalence classes. For every $l \leq 6k + 6$, and for every connected l-terminal graph G with at least $r + 1$, and at most $18(r + 1)(k + 1)$ vertices, add the rule $G \to G_Q$ to a set of reduction rules $\mathcal{A}$, (G_Q the representing terminal graph of the equivalence class to which G belongs.) Similar as in theorem 5, one can prove that the resulting set $\mathcal{A}$ is finite, safe, complete and decreasing. Lemma 4 shows that there exist at least $\frac{|V|}{36(k+1)(r+1)}$ independent applications of a reduction rule in A, when $|V| \geq r+1$. □

To transform a set of reduction rules (or reduction-counter rules) as described in theorem 7 into a parallel algorithm, we first must have a method to find in parallel a large enough set of independent applications of applicable rules. At present, the only methods we know of yield slower parallel algorithms than other presently known parallel algorithms to solve problems on graphs with bounded treewidth. However, when we impose a degree bound on the input graph, then the following method works, and compares favorably with existing solutions.

Our algorithm uses $O(\log n)$ expected time, and $O(n)$ processors on an EREW-PRAM. (Note that the algorithm is randomized; it never produces a wrong answer.) We describe the algorithm for finite index properties. In case of finite integer index, some additional counting must be done in steps 5 and 6 of the algorithm.

Let $G = (V, E)$ be the input graph. Let n be the smallest power of 2, that is at least $|V|$. We have $2n - 1$ processors, n of which may represent a vertex, (the *vertex processors*) and $n - 1$ of which are used for counting purposes (the *counting processors*). The processors are numbered $p_1, \ldots, p_{2n-1}$, with $p_1, \ldots, p_{n-1}$ the counting processors. Suppose we have a fixed set $\mathcal{A}$ of reduction rules (as in theorem 7), each rewriting a subgraph with at least c_1 vertices and at most c_2 vertices to a smaller subgraph. $\mathcal{A}$ is safe for the property $P(G) \wedge tw(G) \leq k \wedge d(G) \leq d$. If a graph with n' vertices belongs to the class, recognized by $\mathcal{A}$, then there exists a concurrent set of reduction rules, such that at least $c_3 \cdot n'$ vertices are an inner vertex of a rule in the set, or $n' \leq c_4$. ($c_1,\ c_2,\ c_4 \in \mathbf{N}$; $c_3 \in \mathbf{R}^+$.) Vertex processors may be alive (when the processor that they represent still is in the reduced graph) or dead (when they do (no longer) represent a vertex.)

The algorithm consists of a number of rounds; each round takes constant time. In each round, the following steps are successively done:

1. Each vertex processor, that is alive, checks whether there exists an applicable reduction rule that has the vertex represented by the processor as inner vertex. If so, the processor is called *applicable* in this round. (This step can be done in constant time, as the degree of the graph is bounded by a constant.)
2. Define the confict graph as follows: each applicable processor forms a vertex in the confict graph. There is an edge between two applicable processors, if at least one the corresponding rule applications contains a vertex of the other processor as inner vertex. Compute the conflict graph. (Note that there is a constant upper bound on the degree of vertices in the conflict graph, say d'. E.g., one can take $d' = d^{c_2}$.)
3. Each applicable processor picks a random number in $\{1, \ldots, d'\}$. It *wins* in this round, if its number is higher than the number, picked by all other vertices. (It follows from well established results on parallel algorithms, that with high probability, at least a constant fraction of the processors wins. Note that applications of reduction rules, corresponding to winning processors are concurrent.)
4. Carry out all applications of reduction rules corresponding to winning processors. Each new vertex in a right hand side of a rule in this set is given to a processor

that owned a (now obsolete) vertex in the left hand side of this rule, such that no processor receives more than one vertex. Note that with high probability, a constant fraction of the processors lose the vertex they owned and do not get a new vertex: they die, and do not perform any steps anymore.

5. Each vertex processor p_i sets $al_i = 1$, if p_i is alive, and $al_i = 0$, if p_i is dead. It sets $ap_i = 1$, if p_i is applicable, and $ap_i = 0$, if p_i is not applicable.
6. Each counting processor p_i executes:
   $al_i := al_{2i} + al_{2i+1}$;
   $ap_i := ap_{2i} + ap_{2i+1}$.
   If the round number is at least $\log n$, then al_1 (ap_1) denotes the number of processors, that were alive (applicable) $\log n$ rounds ago. Processor p_1 does the following steps: Check whether $ap_1 \geq c_3 \cdot al_1$. If this does not hold, then we can conclude that the input graph does not belong to the class to be recognized (too little concurrent rule applications were possible). In that case, we stop. Otherwise, check whether $al_1 \leq c_4$. If so, the remaining graph is of constant size: solve the problem in $O(1)$ time.

Note: if the graph we deal with is in the class to be recognized, then at least a constant fraction of the alive processors is applicable. With high probability, at least a constant fraction of these wins, and hence with high probability, at least a constant fraction of the active processors dies in the round. Standard counting arguments show that the average number of rounds is $O(\log n)$.

Theorem 8. *Let k, d, be constants. Let P (R) be finite (integer) index. Then there exists a randomized algorithm, that solves P (R) on graphs G with treewidth at most k and maximum degree at most d in $O(\log n)$ expected time on an EREW-PRAM with $O(n)$ processors.*

Corollary 9. *For constant k, d, the class of graphs with treewidth $\leq k$ and maximum vertex degree $\leq d$ is recognizable in $O(\log n)$ expected time on an EREW-PRAM with $O(n)$ processors.*

Using similar techniques one can also obtain sequential algorithms using graph reduction which need only linear space and time.

Theorem 10. *If for property P there exists a finite, safe, complete, and decreasing set of reduction rules $\mathcal{A}$ for P, such that all terminal graphs in a left-hand-side of a rule in $\mathcal{A}$ are connected, and there exist constants $c \in \mathbf{R}^+$, $c' \in \mathbf{N}$ such that for all graphs $G = (V, E) \in P$, there exists a concurrent set of $c \cdot |V|$ applications of rules from $\mathcal{A}$, or $|V| \leq c'$, then there exists a recognition algorithm for P that uses linear time and linear space.*

6 Final remarks

6.1 Constructing solutions

The algorithms discussed so far are decision algorithms: no solutions are constructed. For instance, we have established an algorithm to determine the size of a maximum independent set of a graph with bounded treewidth, but this algorithm does not yield the independent set of maximum size itself. In several cases, it is possible to solve the construction versions of problems, by first performing the reduction algorithm (as described before in this paper) and then undoing the reduction steps one by one, while keeping a solution of the graph that we are working with.

Lemma 11. *Suppose $H_2 \rightarrow^{+i} H_1$ is a safe reduction-counter rule for the Independent Set problem, $H_1 = (V_1, E_1)$, $H_2 = (V_2, E_2)$ l-terminal graphs. There exists a mapping $f : \mathcal{P}(V_1) \rightarrow \mathcal{P}(V_2)$, such that for all l-terminal graphs G, if S is a maximum independent set in $H_1 \oplus G$, then $S - (S \cap V_1) \cup f(S \cap V_1)$ is a maximum independent set in $H_2 \oplus G$.*

It follows that when we undo the reduction step $H_2 \rightarrow^{+i} H_1$, we can compute the solution for the independent set problem for the larger graph from the solution for the smaller graph in constant time: we need only to look at how the solution looks like in the part of the graph that is rewritten. In total, construction of the solution for the original input graph takes $O(n)$ time. The same approach can be applied to the parallel algorithms of section 5.

It seems that this approach works for many other problems: Hamiltonian circuit, graph coloring, vertex cover, etc. We conjecture that there are general descriptions of classes of problems that can be dealt with in this way. An interesting candidate for such a class would be the class of problems that can be expressed in monadic second order logic (see e.g. [8]). Another interesting open problem is whether it is possible to find tree-decompositions and/or pathdecompositions with small treewidth or pathwidth with this approach.

6.2 Graph reduction as an heuristic to solve problems on arbitrary graphs

Graph reduction may possibly also be a useful tool which helps to solve hard problems on arbitrary (sparse) graphs. A possible approach to many such problems could be:

- Fix some safe and decreasing set of reduction or reduction-counter rules.
- Apply reduction rules on the input graph and reduced versions, until no rule application is possible.
- Use another method to solve the problem on the graph after the reductions.
- Possibly, construct a solution for the original problem from the solution for the reduced problem by using the method of section 6.1.

The hope is, of course, that the graph after the reductions is smaller than the original graph, and hence, that the running time of the third step is much smaller than the running time when this algorithm would have been applied directly to the input graph. Provided that the reductions can be carried out quick enough, this approach may be a nice tool to reduce the time needed to solve some graph problems on arbitrary sparse graphs.

6.3 Open problems

This paper leaves several directions for further research open. Are there general characterizations of large classes of finite integer index problems? What classes of graphs (without any bound on the treewidth) can be recognized using graph reduction? For instance, the class of graphs with maximum vertex degree d (d fixed) is recognizable with graph reduction. Are there other, interesting classes of graphs with this properties? Is the class of planar graphs recognizable in this way? Classes of graphs with bounded genus? (The latter problem seems particular interesting, as this may be an approach to obtain much faster algorithms for recognition of graphs with bounded genus than known so far.) Can we find efficient parallel algorithms for graphs with bounded treewidth without a degree bound? Is it possible to decrease the number of processors used in the algorithm of section 5 to $O(n/\log n)$?

Acknowledgement

The author thanks Ton Kloks and Hans Zantema for useful discussions and remarks on the subject of this paper.

References

1. K. R. Abrahamson and M. R. Fellows. Finite automata, bounded treewidth and well-quasiordering. In *Graph Structure Theory, Contemporary Mathematics vol. 147*, pages 539–564. American Mathematical Society, 1993.
2. S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. In H. Ehrig, H. Kreowski, and G. Rozenberg, editors, *Proceedings of the Fourth Workshop on Graph Grammars and Their Applications to Computer Science*, pages 70–83. Springer Verlag, Lecture Notes in Computer Science, vol. 532, 1991. To appear in J. ACM.
3. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340, 1991.
4. H. L. Bodlaender. NC-algorithms for graphs with small treewidth. In J. van Leeuwen, editor, *Proc. Workshop on Graph-Theoretic Concepts in Computer Science WG'88*, pages 1–10. Springer Verlag, Lecture Notes in Computer Science, vol. 344, 1988.
5. H. L. Bodlaender. A tourist guide through treewidth. Technical Report RUU-CS-92-12, Department of Computer Science, Utrecht University, Utrecht, 1992. To appear in Acta Cybernetica.
6. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 226–234. ACM Press, 1993.
7. B. Courcelle. Graph rewriting: an algebraic and logical approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B*, pages 192–242, Amsterdam, 1990. North Holland Publ. Comp.
8. B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.
9. B. Reed. Finding approximate separators and computing tree-width quickly. In *Proceedings of the 24th Annual Symposium on Theory of Computing*, pages 221–228, 1992.
10. N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.