Efficient Parallel Algorithms for Tree-Decomposition and Related Problems *

Jens Lagergren[†]

Abstract

We present an efficient parallel algorithm for the tree-decomposition problem for fixed width w. The algorithm runs in time $O(\log^3 n)$ and uses $O(n)$ processors on a CRCW PRAM. This enables us to construct efficient parallel algorithms for three important classes of problems: MS properties, linear EMS extremum problems and enumeration problems for MS properties, when restricted to graphs of tree-width at most w.

We also improve the sequential time complexity of the tree-decomposition problem for fixed w and state some implications of this improvement.

1 Introduction

When characterizing graph problems that admit efficient parallel algorithms it is reasonable to avoid NP complete problems. With this in view two parameters can be varied: the generality of the characterization and the class of graphs considered. We choose the latter and restrict ourselfs to graphs of tree-width at most w. Intuitively, graphs of tree-width at most w are those that can be constructed by piecing together graphs of size $w+1$, at vertex sets of size w, in a tree like fashion. A tree-decomposition can be described as a tree that guides this construction.

Basically two arguments support this choice of restriction. A very general characterisation based on classical logic is achieved. Many interesting and well studied classes of graphs that are often encountered in applications have a universal bound for the tree-width of their members. As an example, trees and forests have tree-width at most 1, series-parallel graphs and outerplanar graphs at most 2, almost trees with w additional edges have tree-width at most $w+1$, Halin graphs at most 3, and k-terminal graphs have tree-width at most k. Furthermore partial k-trees are exactly the graphs of tree-width at most k.

The reason for our interest in the tree-decomposition problem is that it is the absolute bottleneck to achieve the characterisation we aim for.

In section 9 we list a large number of problems that can be solved efficiently in parallel with our technique, when restricted to the class of graphs of tree-width at most w. Almost all of these problems are NP-complete for the class of all graphs, we list them with references to Garey and Johnson [10].

The model of parallel computations we work with is the arbitrary CRCW PRAM (common read common write parallel random access machine). There are two quantities of fundamental importance connected to a parallel algorithm, the running time, $t(n)$, and the number of processor used, $p(n)$, on an instance of size n. The product of these two, $W(n) = t(n)p(n)$, is called the *work*. Assume that there is a parallel algorithm for a problem S that performs $W(n)$ work and has running time $t(n)$. Then the parallel algorithm is called *efficient* if $W(n) = T(n)q_1(\log n)$ and $t(n) = q_2(\log n)$, where $T(n)$ is the running time of the best sequential algorithm for S, and q_1 and q_2 are two polynomials, see [13].

Since there are also other PRAM models we would like to stress the fact that the notion of efficiency is invariant with respect to these models, see [13].

Given a graph G of tree-width at most w our parallel algorithm produces a tree-decomposition of G of depth at most $O(\log n)$ and width at most $6w+5$. It runs in time $O(\log^3 n)$ and uses $O(n)$ processors. Hence the trivial lower bound $\Omega(n)$ on the complexity of a sequential algorithm for this problem implies the efficiency of our parallel algorithm.

The only previously known parallel tree-decomposition algorithms are due to Bodlaender [6] and Chandrasekharan *et al.* [9]. Boedleander's algorithm runs in time $O(\log n)$ and uses $O(n^{3w+4})$ processors. The algorithm of Chandrasekharan *et al.* runs in time $O(\log^2 n)$ and uses $O(n^{2w+5})$ processors. Thus both these algorithms have better running times than ours. They also have the advantage

*Supported by the Swedish Board for Technical Development.

[†]Department of Numerical Analysis and Computing Science, The Royal Institute of Technology, S-100 44 Stockholm, Sweden.

that they construct a tree-decomposition of width w. The number of processors needed is however far from satisfactory in both cases. None of the two are efficient since the work performed by the least costly is $O(n^{2w+5}\log^2 n)$ and it is most unfortunate that the degree of this polynomial grows with w.

The first sequential tree-decomposition algorithm by Arnborg *et al.* [2] is based on dynamic programming and has running time $O(n^{w+2})$. In [17] Robertson and Seymour, among many other things, improve this result when they describe a $O(n^2)$ tree-decomposition algorithm. The latter algorithm is the best sequential algorithm previously known for this problem. Thus the sequential variation of our tree-decomposition algorithm compares favorably. In fact a closer look at the implementation of the sequential variation of our algorithm reveals that it runs in time $O(n\log^2 n)$.

Our parallel tree-decomposition algorithm consists essentially of the main algorithm and a subalgorithm called the ***nice separator algorithm***. The nice separator algorithm finds, when applied to a graph G of tree-width at most w, a separator that is nice in the sense that it splits the graph G into "small" components.

The main algorithm is a fairly straightforward parallel extension of Robertson's and Seymour's sequential tree-decomposition algorithm.

The difficult part of our algorithm is the nice separator algorithm. We know of no previous algorithm other than the brute force method for this problem. The work performed by our nice separator algorithm is $O(n\log^2 n)$, which should be compared to the $O(n^{w+2}\log n)$ work performed by the brute force method.

The characterizations of problems admiting efficient parallel solutions that we will consider are based on MS (monadic second order) logic. The MS language contains all the symbols of the first order language as well as the membership symbol $\in$ and set variables $X_1, X_2, \ldots$ (denoted by uppercase letters as opposed to individual variables). Quantification, both universal and existential, over these set variables is allowed. The main applications of our parallel tree-decomposition algorithm are efficient parallel algorithms for deciding MS properties, solving linear EMS extremum problems and solving enumeration problems for MS properties, restricted to graphs of tree-width at most w. MS properties are properties expressible in the monadic second order logic language. Linear EMS extremum problems (linear extended monadic second order extremum problems) are problems where we want to maximize or minimize a linear function over sets definable in the monadic second order logic. Enumeration problems for MS properties are problems where we want to know how many sets there are that satisfy a given monadic second order language sentence.

For trees have characterization of problems that admit efficient parallel algorithms been given earlier. In [12] He and Yesha showed that BTAC (binary tree algebraic computation) problems, which is closly related to MS properties for trees, can be solved efficently. Abrahamson *et al.* improved this in [1]. They prove that optimal algorithms are possible to obtain for an extension of the class of BTAC problems.

Series parallel graphs have also been considered [11, 12].

2 Definitions

A ***graph*** G is a set $V(G)$ of ***vertices*** and a set $E(G) \subseteq V(G)\times V(G) - \{(v,v)|v \in V(G)\}$ of ***edges***. A graph G is said to be connected if there is a ***u, v-path***, a path from u to v, for each pair of vertices u, v in G. Every maximal connected subgraph of a graph G is called a ***connected component*** of G.

The ***subgraph*** of G ***induced*** by a set $W \subseteq V(G)$ is the graph $G[W] = G'$, where $V(G') = W$ and $E(G') = \{(v,u) \in E(G)|v,u \in W\}$. By $G - S$, where $S \subseteq V(G)$, the graph $G[V(G)-S]$ is meant. If $G-S$ is not connected then S is called a ***separator***. Moreover $C(u,S)$ is used to denote the connected component in $G - S$ that contains u, whenever $u \in V(G) - S$.

A ***c-nice*** separator in a graph G with n vertices is a separator S such that no connected component in $G - S$ has more than $(1-c)n$ vertices.

By the ***depth*** of a tree T we mean the maximum distance between two vertices in T. This is sometimes called the ***diameter***.

We denote the set $\{1,\ldots,n\}$ by $[n]$. A ***k-set*** is a set of cardinality k. For other graph theoretic definitions see [7] or [8].

The concept of tree-width, defined below, was introduced by Robertson and Seymour in [16].

Definition 2.1 *Let G be a graph. A tree-decomposition of G is a pair (X,T), where T is a tree and X is a family of subsets of $V(G)$ indexed by $V(T)$, such that:*

- $\bigcup_{t\in V(T)} X_t = V(G)$
- *for every edge $(v,u) \in E(G)$ there is a $t \in V(T)$ such that $v, u \in X_t$*

- *if t' lies on a path from t to t'' in T then $X_t \cap X_{t''} \subseteq X_{t'}$.*

The width *of a tree-decomposition (X,T) is $\max_{t \in V(T)} |X_t| - 1$. A graph has* tree-width at most *w if there is some tree-decomposition of G of width at most w.*

The *depth* of a tree-decomposition (X,T) is the depth of T.

Remark: It is easy to see that graphs of tree-width at most w have $O(n)$ edges. This fact will be used throughout the paper with no further comments.

3 Separators

Our main algorithm, explained in section 8, finds separators of two kinds. The first kind is separators as in Theorem 3.1 below, where Z will be a set of cardinality bounded by a constant. This kind will not cause any problem. They can be found using standard flow techniques. The second kind, the nice separators, will be harder to find. In fact to describe a way to find such separators, *i.e.* the nice separator algorithm, will occupy us until section 7. The first major step in this direction is Theorem 3.2 that predicts the existence of a path separator or a star separator in graphs of tree-width at most w. Path and star separators are defined in the text following the theorem. There we also try to give an intuitive idea of why path and star separators are easier to find than just any nice separator.

The following was proved in [16, Theorem 2.6].

Theorem 3.1 (Robertson and Seymour) *Let G be a graph of tree-width at most w and Z a subset of $V(G)$. Then there exists a separator S in G of cardinality at most $w+1$ such that if C is a connected component in $G - S$ then the following holds*

$$|V(C) \cap Z| \leq \frac{1}{2}|Z - S|.$$

In the following theorem a function $c_0(x) = 2^{-(x+2)}/(x+1)$ is used for the first time. It will be used throughout the paper and even if its exact definition often is irrelevant we advice the reader to remember that for $x \geq 0$ is $c_0(x)$ a number in $(0,1)$.

Theorem 3.2 *Let G be a graph of tree-width at most w and with $n \geq 4(w+1)$ vertices. Then there exists a $c_0(w+1)$-nice separator S in G such that the following holds for a set M of connected components in $G - S$: the separator S is a minimal u,v-separator for each pair of vertices u, v from different connected components in M, and S and M satisfy either (a) or (b).*

(a) $|M| = 2$ and $|V(C)| \geq c_0(w+1)n$ for $C \in M$.

(b) If $M' \subseteq M$ such that $|M'| \leq w+1$ then $|\bigcup_{C \in M - M'} V(C)| \geq c_0(w+1)n$.

Consider trees (*i.e.* connected graphs of tree-width at most 1). The typical tree, T, for which there exist an S and an M that satisfy (a) in the above theorem is a tree with high diameter, *i.e.* somewhere in T there is a long path. The separator, S, will then be a singleton set, where its single member is a vertex from somewhere in the middle of the path. The typical tree T for which there exist an S and an M that satisfy (b) in the above theorem is a tree T with low diameter, *i.e.* somewhere in T there is a star-like configuration. The separator S will be a singleton set in this case to, but here its single member is the center of the star-like configuration.

Because of this S will be called a ***path separator*** when S and M satisfy Theorem 3.2 by (a), and a ***star separator*** when S and M satisfy Theorem 3.2 by (b). In both cases M will be called a ***set of associated components.*** We will say that S is a path separator or a star separator ***with respect to*** u and v whenever u and v belong to different components in M. When there is a star or path separator with respect to two vertices u and v then the pair u, v will be called a ***good pair***, and both u and v will be called ***good vertices.***

Now let us return to our example and consider trees again. Given a good pair u, v in a tree T, how shall we find, say, a $c_0(w)/2$-nice separator? Let us see what happens if we just pick an arbitrary vertex y on the unique u,v-path in T. If $\{y\}$ is a $c_0(w)/2$-nice separator then we are satisfied. Otherwise if $\{x\}$ is the path separator or the star separator with respect to u and v either u or v, say u, will belong to a "large" connected component in $T - \{y\}$. From this it will be possible to conclude that x is somewhere between y and u on P.

This example tells us that given a good pair some sort of binary search could be an appropriate way to find a $c_0(w)/2$-nice separator. Of course, for tree-width greater than 1 matters will become more complicated but as we will see, this is a possible way to proceed.

4 Minimal u,v-separators

To be able to formulate an algorithm based on binary search we need some notion of before and after.

Therefore we will devote this section to show how some well known combinatorial structures can be defined on the set of minimal u, v-separators in a graph G.

If u and v are two vertices of a connected graph G, and S and S' two minimal u, v-separators then it seems natural to say that S comes before S' seen from u if no u, S'-path avoids S. This actually gives a partial order on the family of minimal u, v-separators. That is, if we fix a pair of vertices u, v and their internal order then a partial order is induced on the family of minimal u, v-separators, F; if $S, S' \in F$ then $S \leq_{u,v} S'$ *iff* no u, S'-path avoids S. We will sometimes use terminology involving phrases like: S comes before S' seen from u ($S \leq_{u,v} S'$) and S comes after S' seen from u ($S' \leq_{u,v} S$).

If we as well as the two vertices u, v and their internal order fix a maximal number of vertex disjoint u, v-paths, say $P_1, \ldots, P_k$, and let K be the family of all k-sets (sets of cardinality k) that contain exactly one internal vertex of each of the paths P_i, *i.e.*

$$K = \{X | X \subseteq \bigcup_{i=1}^{k} V(P_i) - \{u, v\}, |X \cap P_i| = 1\},$$

then this also induces a lattice on K. To see this, notice that each path P_i has a natural total order on its internal vertices defined by: $x \leq y$ *iff* the unique u, y-subpath of P contains x. This means that the internal vertices of each of the paths P_i is a lattice where join ($\frown$) and meet ($\smile$) are defined by: $x \frown y$ is the least element of x and y, $x \smile y$ is the greatest element of x and y respectively.

Let $X(i)$ be the single vertex of $X \cap P_i$ for each $i \in [k]$ and $X \in K$. Define $X \frown Y$ and $X \smile Y$ by: $X \frown Y = \{X(i) \frown Y(i) | i \in [k]\}$ and $X \smile Y = \{X(i) \smile Y(i) | i \in [k]\}$. It is not hard to prove that the above definitions of meet and join give a lattice on K. Actually, this is the cardinal product as defined in [5].

Another useful observation is that the family of minimum u, v-separators (*i.e.* u, v-separators of cardinality k) is a sublattice of K.

5 Surrounding a separator

To simplify the formulation of the rest of the theorems in this paper we define c_0 to be $c_0(w)$ and c_1 to be $c_0/2$. We would also like to make the following remark and definition. Sometimes when we have two vertices v_1, v_2 of a graph G we will need to make v_1 adjacent to each vertex in the neighborhood of v_2. We therefore introduce $G_{v_1}^{v_2}$ to denote this, *i.e.* $G_{v_1}^{v_2} = G \cup \{(v_1, v_3) | v_3 \in N_G(v_2)\}$.

Assume that S is a path separator or a star separator with respect to u and v. Given this good pair our binary search algorithm will in principle look as follows. We have two minimal u, v-separators S_u and S_v, where S_u comes before S and S comes before S_v seen from u, such that $V(C(u, S_u))$ and $V(C(v, S_v))$ are small components (have less than $c_1 n$ vertices). Then we choose a minimal u, v-separator S' as close to "the middle" of S_u and S_v as possible. If S' is c_1-nice then we are satisfied. If $V(C(u, S'))$ is small (has less than $c_1 n$ vertices) then we set $S_u = S'$ and if $V(C(v, S'))$ is small then we set $S_v = S'$.

In this way the number of vertices between S_u and S_v will decrease exponentially in the number of steps and the hope is of course that S will continue to lie between S_u and S_v. This is actually true when S is a star separator, see Lemma 5.3, but it is too much to hope for when S is a path separator. Instead we then have to settle with a lemma that states that some part of S have to lie between S_u and S_v, see Lemma 5.2.

However when the u, v-connectivity is equal to the cardinality of S, another c_1-nice separator S_0 can be constructed from S_u, S_v, and the part of S that lies between S_u and S_v, see Lemma 5.1. There is also another complication when the u, v-connectivity is not equal to the cardinality of S.

We are only able to find minimum separators. Hence if S_u and S_v have cardinality less than that of S and there is no minimum separator between S_u and S_v the problem of how to proceed arises.

The way we solve these two problems is to notice that if at least some part of S is between S_u and S_v then there is some $x \in S_u$ and some $y \in S_v$ such that $(G_u^x)_v^y$ not only has greater u, v-connectivity than G but also has S as a path separator or a star separator.

Lemma 5.1 *Let S_u and S_v be two minimum u, v-separators of cardinality k, in a graph G with n vertices, that satisfy $|V(C(u, S_u))| < c_1 n$ and $|V(C(v, S_v))| < c_1 n$ respectively. Assume that there is a path separator S, with respect to u, v, of cardinality k in G. Then there is a c_1-nice separator S_0 of cardinality k between S_u and S_v.*

Lemma 5.2 *Let S be a path separator in G, with respect to u and v. Assume that S' is a minimal u, v-separator that is not c_0-nice. Then* $S \not\leq_{u,v} S'$ if $|V(C(u, S'))| \leq c_0 n$ and $S' \not\leq_{u,v} S$ if $|V(C(v, S'))| \leq c_0 n$.

Lemma 5.3 *Let S be a star separator in G, with respect to u and v. Assume that S' is a minimal u, v-separator that is not c_0-nice. Then*

$S' <_{u,v} S$ if $|V(C(u,S'))| \leq c_0 n$ *and* $S <_{u,v} S'$ if $|V(C(v,S'))| \leq c_0 n$.

Lemma 5.4 *Let S_u and S_v be two minimum u,v-separators of cardinality k none of them c_0-nice, where S_u comes before S_v seen from u, in a graph G. Assume that S_u and S_v satisfy $|V(C(u,S_u))| < c_1 n$ and $|V(C(v,S_v))| < c_1 n$ respectively. Assume also that there is no u,v-separator of cardinality k between S_u and S_v but there is a path separator or a star separator S in G with respect to u and v. Then there is a vertex $x \in S_u$ and a vertex $y \in S_v$ such that the u,v-connectivity in $(G_u^x)_v^y$ is at least $k+1$ and S is a star or path separator in $(G_u^x)_v^y$, with respect to u and v.*

6 When given a good pair

The first algorithm in this section is the binary search algorithm. Given a graph G and a good pair u,v it either outputs a c_1-nice separator or a family of at most w^2 graphs such that all of the graphs have higher u,v-connectivity than G and in at least one of the graphs u,v is a good pair. We call this family a u,v-strengthening of G.

Given a good pair the second algorithm applies the binary search algorithm iteratively until a c_1-nice separator is found.

Algorithm 6.1 *The binary search algorithm.*

- *Inputs:* A graph G and two of its vertices u,v.
- *Output:* To make the description of the output easier assume that the u,v-connectivity in G is k. A c_1-nice separator is always a correct output. If $k < w$ then also a u,v-connectivity strengthening is a correct output. If u,v is not a good pair then the empty set or a u,v-connectivity strengthening is a correct output as well as a c_1-nice separator. If $k > w$ then only the empty set is a correct output.

First some initializations are made.

(1) Find $w+1$ or a maximal number of vertex disjoint u,v-paths. If $w+1$ such paths are found stop and let $\emptyset$ be the output. Otherwise call these paths $P_1,\ldots,P_k$. Let K be the k-set lattice that these paths induces and M the sublattice of K that consists of the u,v-separators of cardinality k.

(2) Set $S_u = 0_M$ and $S_v = I_M$. (Where 0_M and I_M are the bottom respectively the top element of the lattice M.)

Then the following tests are performed before the main loop is executed.

(3) If S_u or S_v is a c_1-nice separator then stop and let it be the output.

(4) If $|V(C(u,S_v))| < c_1 n$ and $|V(C(v,S_v))| < c_1 n$ then stop an let $\emptyset$ be the output.

(5) If $|V(C(u,S_u))| < c_1 n$ and $|V(C(v,S_u))| < c_1 n$ then stop an let $\emptyset$ be the output.

(6) If $|V(C(u,S_v))| < c_1 n$ then stop and let $E = \{G_u^x | x \in S_v\}$ be the output.

(7) If $|V(C(v,S_u))| < c_1 n$ then stop and let $E = \{G_v^x | x \in S_u\}$ be the output if $k < w$ and $\emptyset$ if $k = w$.

After that we repeat steps 8 to 14 until a stop instruction is executed (in step 9,10,11 or 14).

(8) Choose X (of cardinality k) such that $X(i)$ is the vertex in the middle of $S_u(i)$ and $S_v(i)$ on P_i. Chose S' as the greatest separator in M such that $S' \leq_K X$ and choose S'' as the least separator in M such that $S' <_M S''$.

(9) If S' or S'' is a c_1-nice separator then stop and let it be the output.

(10) If $|V(C(u,S''))| < c_1 n$ and $|V(C(v,S''))| < c_1 n$ then stop and let $\emptyset$ be the output.

(11) If $|V(C(u,S'))| < c_1 n$ and $|V(C(v,S'))| < c_1 n$ then stop and let $\emptyset$ be the output.

(12) If $|V(C(u,S''))| < c_1 n$ then set $S_u = S''$

(13) If $|V(C(v,S'))| < c_1 n$ then set $S_v = S'$

(14) If none of the conditions in step 6,7 or 8 are satisfied then stop and let $E = \{(G_v^x)_u^y | x \in S'$ and $y \in S''\}$ be the output if $k < w$ and $\emptyset$ if $k = w$

We will now indicate why this algorithm is correct and why it terminates within $O(\log^2 n)$ steps. We start with its correctness.

There are ten places where a stop instruction possibly can be executed namely at step 1,3,4,5,6,7,9,10,11, and 14. At step 3 and 9 the output is a c_1-nice separator and hence satisfies the algorithm description. In step 1 we only stop if the u,v-connectivity is greater than $w+1$ and then correctly outputs the empty set. If the conditions in step 10 are satisfied (notice that S_v is not c_1-nice). Then, by theorem 5.2 and theorem 5.3, there can not be any path separator or star separator in G, with respect to u, v. Step 4,5 and 11 are analogous.

This leaves step 6,7 and 14. That the output in step 14 is according to the description if $k < w$ follows from Lemma 5.4 if we can prove that there never is any u, v-separator of cardinality k strictly between S_u and S_v when step 14 is executed. This is however immediate since we chose S'' as the *least* separator in M such that $S' <_M S''$. If $k = w$ and there is a path separator or a star separator, with respect to u, v, then Lemma 5.1 and Lemma 5.3 assures that this situation never will occur. Step 6 and 7 are analogous.

Let us now turn to the algorithm's complexity. In each repetition of steps 8-14 there is some i such that the distance between $S_u(i)$ and $S_v(i)$ decreases by a factor 2, since $S' \leq_K X$, $S'' \not\leq X$ and X is in the middle of S_u and S_v. Hence steps 8-14 will be repeated at most $k \log n$ times. To find a maximal number of vertex disjoint paths and separators the algorithm of Khuller and Shieber ([14]) is used. The algorithm runs in $O(\log n)$ time and uses $O(n)$ processors on a CRCW PRAM. No other operation in this algorithm require more resources, see [13].

Note that the above algorithm can be implemented to run in $O(n \log n)$ time on a RAM since the paths and separators then can be found by a standard flow algorithm.

Algorithm 6.2

- *Inputs:* A graph G, and two of its vertices u and v.
- *Output:* If there is a path separator or a star separator S in G of cardinality at most w, with respect to u, v, then the output is a c_1-nice separator S_0. Otherwise either an answer 'NO' or a c_1-nice separator S_0 is output.

The actual algorithm, which uses Algorithm 6.1, is fairly straightforward and has been omitted.

7 How to find a good pair

Given a graph G with a path separator or star separator S, we know how to find a c_1-nice separator in G, but only under that assumption that we already have a good pair. So the question to be answered now is: How shall we proceed to obtain a good pair u, v? One simple method would be to try all pairs of vertices in G. The drawback is of course that then $O(n^2)$ processors must be used to achive polylog running time.

The route we will follow takes advantage of the following fact. A constant fraction of all the vertices in G are good vertices and these good vertices are separated from the rest of the vertices by S. This makes it possible to prove that if T is a spanning tree for G then good vertices and vertices from S will lie together in connected subtrees of T of size $\Omega(n)$. This last observation implies that there is a set A of constant size such that we either can choose a good pair u, v or an $s \in S$ from A. The set A will be a cn-splitter for T, where k-splitters are defined as follows.

Definition 7.1 *Let T be a tree with n vertices and A a subset of the vertex set of T. Then A is called a* k-splitter *for T if A has cardinality at most n/k and all connected components in $T - A$ have less than k vertices.*

To be able to find k-splitters in time $O(\log n)$ with $O(n)$ processors we shall now introduce another type of sets, k threshold sets. Let T be a rooted tree. Define the descendant function of v, $f(v)$, to be the number of descendants of v in T for each $v \in T$. Let $f_k(v)$ be the remainder of $f(v)$ divided by k. Let A be the set defined by

$$A = \{v \in V(T) | \sum_{v' \in \mathrm{Sons}(v)} f_k(v') > k\},$$

where $\mathrm{Sons}(v)$ is the set of sons of v, then A is called the ***k threshold*** set for T. The set A is actually also a k-splitter for the underlying tree. We state this as a theorem.

Theorem 7.2 *Let T be a rooted tree with n vertices and A the k threshold set for T. Then A is a k-splitter for T.*

A k threshold set, and hence also a k-splitter, is easy to find using expression evaluation techniques described in [15, Theorem 15].

Theorem 7.3 *If G is a graph with a path separator or a star separator S (where $|S| = s$), T is a spanning tree for G and A is a $\frac{c_2}{s}n$-splitter for T, then (i) or (ii) holds.*

(i) There are u and v in A such that S is a path or star separator with respect to u and v.

(ii) $S \cap A \neq \emptyset$

Algorithm 7.4 *The nice separator algorithm.*

- *Input:* A graph G.

- *Output:* A nice separator if G has a path separator or a star separator of cardinality at most w. Otherwise either a nice separator or the answer 'NO'.

The actual algorithm, which uses Algorithm 6.2, is fairly straightforward and has been omitted.

8 The Main Algorithm

We are now about to present the main algorithm, the parallel extension of Robertson's and Seymour's sequential tree-decomposition algorithm, see [17]. The reader familiar with the latter will notice the similarities even though Robertson's and Seymour's algorithm was formulated in terms of branch-width in [17].

Algorithm 8.1 *The tree-decomposition algorithm for width w.*

- *Input:* A graph G.
- *Output:* If G has tree-width at most w a tree-decomposition of G of width at most $6w + 5$. If G has tree-width greater than w a tree-decomposition of G of width at most $6w + 5$ or an answer 'NO'.

We describe the algorithm as a way to iteratively modify a tree-decomposition (X, T), where $X = \{X_t | t \in V(T)\}$, of G until the width of the tree-decomposition becomes $\leq 6w+5$. Let us call a vertex $t \in V(T)$ ***big*** if $|X_t| > 6w + 5$. As the algorithm proceeds three conditions are supposed to be satisfied:

(i) (X, T) is a tree-decomposition.

(ii) Only leaves are big in T.

(iii) If t is a big leaf and t' its single neighbor in T then $|X_t \cap X_{t'}| \leq 4(w + 1)$.

First some initializations are made.

(1) Let T be a tree with just one vertex, say, t and X be the family of sets $\{X_t\}$, where $X_t = V(G)$.

Then the following steps are iterated.

(2) If there are no big vertices in T then (since the tree-decomposition has width $\leq 6w+5$) stop and let (X, T) be the output.

(3) In parallel for each big leaf t. Let t' be the neighbor of t and $Z = X_t \cap X_{t'}$. Find a nice separator S_1 with algorithm 7.4 and a separator S_2 as in Theorem 3.1, with respect to Z, in the graph $H = G[X_t]$. If any of these two separators can not be found stop and output 'NO'. Otherwise let $S = S_1 \cup S_2$.

(4) For each component C_j in $H - S$ modify T by creating a new leaf t_j adjacent to t also add a new set X_{t_j} to X. Set $X_t = Z \cup S$ and $X_{t_j} = V(C_j) \cup S$.

Actually, when $|X_t| \leq (4w + 2)/c_0$ holds for a big leaf t then we have to find a separator that satisfies Theorem 3.1, with $Z = \emptyset$. Since the input size then is bounded, this can be done in constant time by a table lookup. It is not hard to prove that each iteration then decreases the width of (X, T) by a constant factor. This implies that the algorithm terminates within $O(\log n)$ iterations and that the depth of T is $O(\log n)$. This tree-decomposition can easily be converted to a tree-decomposition (X', T'), where T' is binary, has depth $O(log^2 n)$ and $O(n)$ vertices, in time $O(logn)$ using $O(n)$ processors.

We end this section by noting that our previous results implies that this algorithm runs in time $O(\log^3 n)$, using $O(n)$ processors, on a CRCW PRAM and in time $O(n \log^2 n)$ on a RAM.

9 Applications

In [4] it is shown, constructively, that for each MS (monadic second order logic) formula φ and number w there is a tree-automaton M and a linear time algorithm $\mathcal{A}$ satisfying the following. Given a tree-decomposition (X, T) of width at most w of a graph G the algorithm $\mathcal{A}$ gives as output a tree T' with $O(n)$ vertices and depth $O(d)$, where n and d are the number of vertices and the depth of T respectively. This tree T' is such that it is accepted by M ***iff*** G has the property described by φ, *i.e.* G is a model for φ. This clearly gives linear time algorithms for graphs of tree-width at most w ***under the assumption*** that they are given with a tree-decomposition of width at most w.

The main application of our parallel tree-decomposition algorithm is to give efficient parallel algorithms for deciding MS properties and solving EMS problems, restricted to graphs of tree-width at most w. We do not give any actual implementation on a PRAM. Instead we firstly note that the tree-automatons used in [4] to decide MS properties, restricted to graphs of tree-width at most w are quite simple. Viewed as a model of parallel computation they are as simple compared to the PRAM as finite state automatons are compared to the RAM. They always imply PRAM algorithms runing in time d and using n processors, where d is the depth of the input tree and n is it's number of vertices. To equip the tree-automatons with counters, as is done in [4] to solve linear EMS extremum problems and enumeration problems for MS properties, does not change the complexity of the PRAM algorithm.

Secondly we note that the algorithm $\mathcal{A}$ easily can be implemented to run on a PRAM in time $O(d)$ using $O(n)$ processors. This clearly gives PRAM algorithms running in time $O(d)$ using $O(n)$ processors for graphs of tree-width at most w *under the assumption* that they are given with a tree-decomposition of width at most w and depth at most d. Since our tree-decomposition algorithm produces tree-decompositions of depth $O(\log^2 n)$ we get the following theorems.

Theorem 9.1 *MS properties can be decided in time $O(\log^3 n)$ on a PRAM using $O(n)$ processors. Moreover, MS properties can be decided in time $O(n \log^2 n)$ on a RAM.*

The latter part of the above Theorem is not a new result. In [3] it is shown that MS properties can be decided in linear time on a RAM that uses polynomially, but not linearly, bounded memory[1]. There it is also shown that this can be done in linear space and time $O(n \log n)$.

In both of the theorems below also the sequential result *is* new.

Theorem 9.2 *Linear EMS extremum problems can be solved in time $O(\log^3 n)$ on a PRAM using $O(n)$ processors. Moreover, linear EMS extremum problems can be solved in time $O(n \log^2 n)$ on a RAM.*

Theorem 9.3 *Enumeration problems for MS properties can be solved in time $O(\log^3 n)$ on a PRAM using $O(n)$ processors. Moreover, enumeration problems for MS properties can be solved in time $O(n \log^2 n)$ on a RAM.*

To show the significance of our results we state two theorems that list a number of MS properties and linear EMS extremum problems respectively. For each of the MS properties there is a corresponding enumeration problem. For example consider the MS property cubic subgraph [GT32]. Corresponding to it is the enumeration problem: how many different cubic subgraphs are there. For a better understanding of why these problems are MS properties, enumeration problems for MS properties and linear EMS extremum problems respectively see [4].

[1]The reason for this apperent paradox is that the algorithm does not use all of its memory but needs a large address space

Theorem 9.4 (Arnborg, Lagergren, Seese) *The following properties are MS properties: domatic number for fixed K [GT3], graph K-colorability for fixed K [GT4], achromatic number for fixed K [GT5], monochromatic triangle [GT6], partition into triangles [GT11], partition into isomorphic subgraphs for fixed connected H [GT12], partition into Hamiltonian subgraphs [GT13], partition into forests for fixed K [GT14], partition into cliques for fixed K [GT15], partition into perfect matchings for fixed K [GT16], covering by cliques for fixed K [GT17], covering by complete bipartite subgraphs for fixed K [GT18], induced subgraph with property P (for monadic second order properties P and fixed K) [GT21], induced connected subgraph with property P (for monadic second order properties P and fixed K) [GT22], induced path for fixed K [GT23], cubic subgraph [GT32], Hamiltonian completion for fixed K [GT34], Hamiltonian circuit [GT37], directed Hamiltonian circuit [GT38], Hamiltonian path (and directed Hamiltonian path) [GT39], subgraph isomorphism for fixed H [GT48], graph contractability for fixed H [GT51], graph homomorphism for fixed H [GT52], path with forbidden pairs for fixed n [GT54], kernel [GT57], degree constrained spanning tree for fixed K [ND1], disjoint connecting paths for fixed K [ND40], chordal graph completion for fixed K, chromatic index for fixed K, spanning tree parity problem, distance d chromatic number for fixed d and k, thickness $\leq K$ for fixed K, membership for each class C of graphs which is closed under minor taking (all of these last problems are unnumbered).*

Theorem 9.5 (Arnborg, Lagergren, Seese) *The following problems are linear EMS extremum problems: vertex cover [GT1], dominating set [GT2], feedback vertex set [GT7], feedback arc set [GT8], partial feedback edge set for fixed maximum cycle length l [GT9], minimum maximal matching [GT10], partition into cliques [GT15], clique [GT19], independent set [GT20], induced subgraph with property Π (for MS property Π) [GT21], induced connected subgraph with property Π (for MS property Π) [GT22], induced path [GT23], balanced complete bipartite subgraph [GT24], bipartite subgraph [GT25], degree-bounded connected subgraph for fixed d [GT26], planar subgraph [GT27], transitive subgraph [GT29], uniconnected subgraph [GT30], minimum K-connected subgraph for fixed K [GT31], minimum equivalent di-*

graph [GT33], Hamiltonian completion [GT34], multiple choice matching for fixed J [GT55], k-closure [GT58], path distinguishers [GT60], maximum leaf spanning tree [ND2], maximum length-bounded disjoint paths for fixed J [ND41], maximum fixed-length disjoint paths for fixed J [ND42], minimum edge(vertex) deletion for any MS property, bounded diameter spanning tree for fixed D [ND4], multiple choice branching for fixed m [ND11], Steiner tree in graphs [ND12], maximum cut [ND16], longest circuit [ND28], longest path [ND29].

In both the above results the problems considered are MS properties or linear EMS extremum problems for the class of all graphs. When the graphs are restricted to have tree-width at most w then it might not be necessary to fix all of the constants that have been fixed above. For instance chromatic number [GT 4] is then a linear EMS extremum problem.

10 Discussion

Our tree-decomposition algorithm makes it possible to prove results similar to the above results for the entire class of EMS problems, defined in [4]. It is possible to prove that for each EMS problem P there is a constant c such that for each w the problem P can be solved by a parallel algorithm running in time $O(\log^3 n)$ using $O(n^c)$ processors, when restricted to graphs of tree-width at most w. What separates this result from what was possible to conclude earlier is that c is a constant *independent* of w.

It follows from [1, Theorem 2.1,Theorem 3.1] that tree automatons imply optimal parallel PRAM algorithms. For trees, where we do not have to find a tree-decomposition, have all MS properties *optimal* algorithms. Hence, the tree-decomposition problem is still a bottleneck. An optimal connected components algorithm would decrease the amount of work performed by the algorithm of Khuller and Shieber by a factor $O(\log n)$, and hence also for our tree-decomposition algorithm. In fact all what is needed is an optimal connected components algorithm for graphs of tree-width at most w.

11 Acknowledgment

We would like to thank Stefan Arnborg, Mikael Goldmann, Joachim Hollman and Johan Håstad for all the very valuable help they have given.

References

[1] K. Abrahamson, N. Dadoun, D.G. Kirkpatrik, and T. Przytycka. A simple parallel tree contraction algorithm. *Journal of Algorithms*, 10:287–302, 1989.

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

[3] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. Technical Report LaBRI-90-02, Laboratorie Bordelais de Recherche en Informatique, 1990.

[4] S. Arnborg, J. Lagergren, and D. Seese. Problems easy for tree-decomposable graphs. *Journal of Algorithms*, to appear.

[5] G. Birkhoff. *Lattice Theory*. American Mathematical Society, 1948.

[6] H.L. Bodlaender. NC-algorithms for graphs with bounded tree-width. Technical Report RUU-CS-88-4, University of Utrecht, 1988.

[7] B. Bollobás. *Graph Theory*. Springer-Verlag, 1985.

[8] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. North-Holland, 1979.

[9] N. Chandrasekharan and A. Hedetniemi. Fast parallel algorithms for tree decomposition and parsing partial k-trees. In *26th Annual Allerton Conference on Communication, Control, and Computing, Urbana-Champaign, Illinois*, 1988.

[10] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, San Francisco, 1979.

[11] X. He. Efficient parallel algorithms for series parallel graphs. Technical Report 8826, University at Buffalo, 1988.

[12] X. He and Y. Yesha. Binary tree algebraic computations and parallel algorithms for simple graphs. *Journal of Algorithms*, 9:92–113, 1988.

[13] R. M. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. Technical Report UCB/CSD 88/408, University of Califonia Berkeley, 1988.

[14] S. Khuller and B. Schieber. Efficient parallel algorithms for testing connectivity and finding disjoint $s-t$ paths in graphs. In *30 th Ann. FOCS*, pages 288–293, 1989.

[15] G. Miller and J. Reif. Parallel tree contraction and its application. In *26 th Ann. FOCS*, pages 478–489, 1985.

[16] N. Robertson and P.D. Seymour. Graph minors ii. algorithmic aspects of tree width. *Journal of Algorithms*, 7:309–322, 1986.

[17] N. Robertson and P.D. Seymour. Graph minors xiii, the disjoint path problem. *Preprint Sept.*, 1986.