JOURNAL OF ALGORITHMS 12, 1–22 (1991)

Algorithms Finding Tree-Decompositions of Graphs

JIŘÍ MATOUŠEK

Department of Computer Science, Charles University, Malostranské nám. 25, 118 00 Prague 1, Czechoslovakia

AND

ROBIN THOMAS*

School of Mathematics, Georgia Institute of Technology, Atlanta, Georgia 30332

Received June 15, 1988; revised December 20, 1989

A graph G has tree-width at most w if it admits a tree-decomposition of width $\leq w$. It is known that once we have a tree-decomposition of a graph G of bounded width, many NP-hard problems can be solved for G in linear time. For $w \leq 3$ we give a linear time algorithm for finding such a decomposition and for a general fixed w we obtain a probabilistic algorithm with execution time $O(n \log^2 n + n \log n |\log p|)$, which for a graph G on n vertices and a real number $p > 0$ either finds a tree-decomposition of width $\leq 6w$ or answers that the tree-width of G is $\geq w$; this second answer may be wrong but with probability at most p. The second result is based on a separator technique which may be of independent interest.
© 1991 Academic Press, Inc.

1. INTRODUCTION

A *graph* is a pair $G = (V, E)$, where V is a finite set and E is a subset of the set of all two-element subsets of V. We write $V(G) = V$ and $E(G) = E$. Our graph-theoretic terminology is standard and is summarized at the end of this section.

A *tree-decomposition* of a graph G is a pair (T, τ), where T is a tree and τ is a mapping from $V(T)$ to the set of all subsets of $V(G)$ such that

(W1) $\bigcup_{t \in V(T)} \tau(t) = V(G)$, and every edge of G has both endpoints in some $\tau(t)$, and

(W2) if $t, t', t'' \in V(T)$, and t' lies on the path from t to t'' in T, then $\tau(t) \cap \tau(t'') \subseteq \tau(t')$.

*This research was carried out at Department of Mathematical Analysis, Charles University, Sokolovská 83, 186 00 Praha 8, Czechoslovakia.

0196-6774/91 $3.00
Copyright © 1991 by Academic Press, Inc.
All rights of reproduction in any form reserved.

The *width* of a tree-decomposition (T, τ) is

$$\max\{|\tau(t)| - 1 : t \in V(T)\},$$

and the *tree-width* of G, denoted by $w(G)$, is the least integer w such that G admits a tree-decomposition of width w. For instance, a graph has tree-width ≤ 1 if and only if it is a forest, it has tree-width ≤ 2 if and only if it is series-parallel, and the complete graph K_n has tree-width $n-1$. Graphs of tree-width $\leq k$ are sometimes called k-decomposable or partial k-trees. We refer the reader to [1–3, 6–9] for more information on tree-width.

Tree-width seems to be a particularly suitable measure of the algorithmic complexity of a graph. Many NP-hard problems can be solved in polynomial or even linear time, provided that we are given a tree-decomposition of G of a width bounded by a constant (see [2, 5, 11]). Hence it is desirable to obtain fast algorithms for finding tree-decompositions of graphs of bounded width. The following is known:

(i) If w is a part of input, the decision problem "is $w(G) \leq w$" is NP-complete [1].

(ii) For fixed w, there is an $O(|V(G)|^{w+2})$ algorithm deciding whether $w(G) \leq w$ and giving a tree-decomposition of G of width w in the positive case [1].

(iii) For every fixed w there exists an $O(|V(G)|^2)$ algorithm deciding whether $w(G) \leq w$ [9]. However, the proof is purely existential (it does not construct the algorithm) and the algorithm does not find a tree-decomposition.

(iv) When we do not insist on the exact determination of $w(G)$, the situation is better: there is a quadratic algorithm which either proves that $w(G) \geq w$ or gives a tree-decomposition of G of width at most $4w$ [11]. Such an "approximate" decomposition suffices for asymptotically fast solution of many NP-hard problems. Let us remark that [11] is written in terms of branch-width, which is a slight variation of tree-width, but this makes only a technical difference.

(v) The problem of (ii) is easy when $w \leq 1$, a linear algorithm when $w = 2$ is described in [4], and an $O(n \log n)$ algorithm when $w = 3$ was found by Arnborg and Proskurowski in [3].

We have a similar algorithm to that of (iv) which is faster, but probabilistic:

1.1. THEOREM. *Let $w \geq 0$ be a fixed integer. There exists a probabilistic algorithm which for a given graph G and a number $p > 0$ produces one of*

(i) *or* (ii) *below in time $O(n(\log n)^2 + n \log n |\log p|)$, where $n = |V(G)|$:*

(i) *a tree-decomposition of G of width at most $6w$,*

(ii) *a (possibly invalid) statement that $w(G) \geq w$.*

For each graph G with $w(G) < w$, the probability that (ii) *is returned is at most p and the result* (i) *is obtained in expected time $O(n \log n)$.*

This result has the following corollaries.

1.2. COROLLARY. *Let $w \geq 0$ be an integer and let $p > 0$ be a real number. There exists a (deterministic) algorithm which given an input graph G on n vertices and with $w(G) \leq w$ determines a tree decomposition of G of width $\leq 6w$. The worst-case running time of this algorithm is $O(n^2)$, and the probability that the running time is $O(n \log n)$ is at least $1 - p$.*

Proof. The algorithm is obtained by running the algorithm from Theorem 1.1 for $cn \log n$ time units, where c is a suitable constant, and if it does not yield the decomposition then switching to the quadratic algorithm of [11]. □

A graph is a *minor* of another if the first can be obtained from a subgraph of the second by contracting edges and deleting loops and multiple edges thus produced. A *lower ideal* is a set $\mathcal{F}$ of graphs with the property that if $H \in \mathcal{F}$ and G is isomorphic to a minor of H then $G \in \mathcal{F}$.

1.3. COROLLARY. *Let $p > 0$ be a real number and let $\mathcal{F}$ be a lower ideal with the property that some planar graph $H \notin \mathcal{F}$. Then there exists a probabilistic algorithm which for a given graph G decides whether $G \in \mathcal{F}$. The worst-case running time of this algorithm is $O(n(\log n)^2)$, the answer "$G \in \mathcal{F}$" is always correct and the answer "$G \notin \mathcal{F}$" may be wrong with probability at most p.*

Proof. By [9] there exists an integer w such that if $w(G) \geq w$ then G has a minor isomorphic to H, and hence $G \notin \mathcal{F}$. By [8] there exists an integer $k \geq 1$ and graphs $H_1, \ldots, H_k$ such that $G \notin \mathcal{F}$ if and only if there exists an i such that $1 \leq i \leq k$ and G has a minor isomorphic to H_i. Our algorithm proceeds as follows:

(1) We apply the algorithm of Theorem 1.1, and if it returns (ii) we return "$G \notin \mathcal{F}$," otherwise we go to Step (2).

(2) From Step (1) we have a tree-decomposition (T, W) of G of width $\leq 6w$. We apply the algorithm of [11] k times to test whether G has a

minor isomorphic to H_i for some i with $1 \le i \le k$. If so we return "$G \notin \mathcal{F}$," otherwise we return "$G \in \mathcal{F}$."

Step (1) takes time $O(n(\log n)^2)$, Step (2) takes time $O(n)$ (indeed, the only step of the algorithm of [11] which takes more than linear time is to find a tree-decomposition of G, but that has been replaced by (1)). The answer returned in Step (1) may be wrong with probability $\le p$ by Theorem 1.1; the answers produced in Step (2) are correct. □

Our second result is an improvement of the algorithm of Arnborg and Proskurowski, as follows.

1.4. THEOREM. *For $w = 1, 2, 3$ there exists an algorithm to decide whether an input graph G on n vertices has tree-width $\le w$, and if so then to construct a tree-decomposition of G of width $\le w$. The algorithm runs in time and space $O(n)$.*

The paper is organized as follows. In Section 2 we develop a separator technique needed for the proof of Theorem 1.1 in Section 3. Theorem 1.4 is proved in Section 4.

Let us introduce some terminology. Let G be a graph and let $A, B \subseteq V(G)$. We say that A, B are *adjacent* (in G) if there is an edge of G with one endpoint in A and the other in B. By $G \setminus A$ we denote the graph obtained from G by deleting vertices of A, and all edges incident with these vertices. Sometimes we will not distinguish between the subset $A \subseteq V(G)$ and the graph induced by this subset. This can cause no confusion. If $A \subseteq V(G)$ induces a connected subgraph then by *contracting A to a* we mean contracting A to a single vertex which will be denoted by a. Paths can have no "repeated" vertices, and if a, b are the endpoints of a path we say that P is a *path from a to b* (or *from b to a*). A *separation* of a graph G is a pair (G_1, G_2) of subgraphs of G such that $V(G_1) \cup V(G_2) = V(G)$, $E(G_1) \cup E(G_2) = E(G)$ and $V(G_1) - V(G_2)$ and $V(G_2) - V(G_1)$ are not adjacent in G. For $v \in V(G)$, the degree of v, denoted by $\deg_G(v)$ is the number of vertices of G adjacent to v.

We need the following proposition whose proof is left to the reader.

1.5. PROPOSITION. *Let G have tree-width $\le w$. Then*

(i) *every minor of G has tree-width $\le w$,*

(ii) *there exists a chordal graph H with $V(H) = V(G)$ and $E(G) \subseteq E(H)$ containing no subgraph isomorphic to K_{w+2},*

(iii) $|E(G)| \le w|V(G)|$.

2. WELL-SPLITTING CUTS

2.1. DEFINITION. Let G be a graph and let a, b be two distinct nonadjacent vertices of G. We define

$$R_0(G,a,b) = \{C \subseteq V(G) : G \setminus C \text{ contains no path from } a \text{ to } b\},$$
$$R(G,a,b) = \{C \in R_0(G,a,b) : \{a,b\} \cap C = \emptyset\},$$
$$k(G,a,b) = \min\{|C| : C \in R(G,a,b)\},$$
$$M(G,a,b) = \{C \in R(G,a,b) : |C| = k(G,a,b)\},$$

and

$$M_0(G,a,b) = M(G,a,b) \cup \{\{a\},\{b\}\}.$$

We call the elements of $R(G,a,b)$ *cuts between a and b* or shortly *cuts*. For $C \in R_0(G,a,b)$ we denote by $\alpha(C)$ ($\beta(C)$, respectively) the set of all vertices $v \in V(G)$ such that there exists a path from v to a (from v to b, respectively) in $G \setminus C$ (so that either $\alpha(C) = \emptyset$ or $a \in \alpha(C)$, and $\alpha(C) \cap \beta(C) = \emptyset$).

2.2. DEFINITION. A *system of Menger paths* (between a and b in G) is a system of paths $P_1, \ldots, P_m$, where each P_i is a path from a to b and $V(P_i) \cap V(P_j) = \{a, b\}$ for $i \ne j$. A *maximum system of Menger paths* is a system of Menger paths $P_1, \ldots, P_k$, where $k = k(G,a,b)$. The vertices of each path in a system of Menger paths are linearly ordered by the relation "u precedes v when passing from a to b." Menger's theorem says that a maximum system of Menger paths exists. By applying the classical Ford–Fulkerson algorithm for finding a maximal network flow one can either find a maximum system of Menger paths, or establish that $k(G,a,b) > m$ in time $O(m|E(G)|)$.

If we interchange the role of a and b in some definition or statement, we obtain a *dual* definition or statement. For example, $\beta(C)$ is a dual notion to $\alpha(C)$. A dual of a valid statement is obviously also valid.

2.3. DEFINITION. (i) Let $C_1, C_2 \in R_0(G,a,b)$. We put $C_1 \vee C_2 = C_1 \cup C_2 \setminus (\alpha(C_1) \cup \alpha(C_2))$ and dually $C_1 \wedge C_2 = C_1 \cup C_2 \setminus (\beta(C_1) \cup \beta(C_2))$.

(ii) We define a relation $\le$ on $M_0(G,a,b)$ by saying that $C_1 \le C_2$ if $\alpha(C_1) \subseteq \alpha(C_2)$.

The following lemma gives some properties of the above defined notions.

2.4. LEMMA. (i) *If* $C_1, C_2 \in R_0(G,a,b)$, *then* $C_1 \vee C_2 \in R_0(G,a,b)$ *and* $\alpha(C_1 \vee C_2) = \alpha(C_1) \cup \alpha(C_2)$.

(ii) *If* $C_1, C_2 \in R_0(G,a,b)$, *then* $\beta(C_1 \vee C_2) \subseteq \beta(C_1) \cap \beta(C_2)$.

(iii) *The relation* $\leq$ *is a partial ordering on* $M_0(G,a,b)$.

(iv) *If* $C_1, C_2 \in M_0(G,a,b)$, *then* $C_1 \leq C_2$ *if and only if* $\beta(C_2) \subseteq \beta(C_1)$.

(v) *If* $C_1 \in M_0(G,a,b)$ *and* $C_2 \in R(G,a,b)$, *then* $|C_1 \vee C_2| \leq |C_2|$.

(vi) $M_0(G,a,b)$ *is a lattice under the ordering* $\leq$, *with lattice operations* $\vee$ *and* $\wedge$, *and* $M(G,a,b)$ *is a sublattice of it. In particular.* $M(G,a,b)$ *has a unique minimal element and a unique maximal element.*

Proof. (i) Clearly $\alpha(C_1) \cup \alpha(C_2)$ is connected or empty. On the other hand, every vertex adjacent to $\alpha(C_1)$ belongs to $\alpha(C_1) \cup C_1$ and similarly for $\alpha(C_2)$; thus every vertex adjacent to $\alpha(C_1) \cup \alpha(C_2)$ belongs to $(C_1 \vee C_2) \cup \alpha(C_1) \cup \alpha(C_2)$, so $\alpha(C_1) \cup \alpha(C_2)$ is either empty or a component of $G \setminus (C_1 \vee C_2)$ and this implies (i).

(ii) By symmetry, it suffices to show $\beta(C_1 \vee C_2) \subseteq \beta(C_1)$. We have $C_1 \subseteq (C_1 \cup C_2) \setminus \alpha(C_1) \setminus \alpha(C_2)) \cup \alpha(C_1) \cup \alpha\ (C_2) = (C_1 \vee C_2) \cup \alpha\ (C_1 \vee C_2), (C_1 \vee C_2) \cap \beta(C_1 \vee C_2) = \emptyset, \alpha(C_1 \vee C_2) \cap \beta(C_1 \vee C_2) = \emptyset$, so $C_1 \cap \beta(C_1 \vee C_2) = \emptyset$. Since $\beta(C_1 \vee C_2)$ is connected, does not meet C_1, and is empty or contains b, it must be contained in $\beta(C_1)$.

(iii) Actually the relation $\leq$ defines a partial ordering on the set of all inclusion-minimal elements of $R_0(G,a,b)$. Since the inclusion is a partial ordering, it suffices to show that if $C \in R_0(G,a,b)$ is inclusion-minimal, then $\alpha(C)$ uniquely determines C. If $\alpha(C) = \emptyset$ then $C = \{a\}$ and if $\alpha(C)$ is adjacent to b then $C = \{b\}$, otherwise C must contain every vertex of $G \setminus \alpha(C)$ adjacent to $\alpha(C)$ and the set of all such vertices forms a cut between a and b, so (by inclusion-minimality of C) C is exactly equal to this set (which is defined in terms of $\alpha(C)$).

(iv) It suffices to show that $\alpha(C_1) \subseteq \alpha(C_2)$ implies $\beta(C_2) \subseteq \beta(C_1)$ (the converse implication follows by duality). The proof is rather similar to (ii). The cases $\alpha(C_1) = \emptyset$ and $C_2 = \{b\}$ are easily treated separately, so in the sequel we assume that neither of them occurs. We know that C_i consists of all vertices of $G \setminus \alpha(C_i)$ adjacent to $\alpha(C_i)$ and so $C_1 \subseteq \alpha(C_2) \cup C_2$, therefore $C_1 \cap \beta(C_2) = \emptyset$ and so $\beta(C_2) \subseteq \beta(C_1)$.

(v) We may suppose that $C_1 \in M(G,a,b)$ (the cases $C_1 = \{a\}$ and $C_1 = \{b\}$ are easy). Let $D = C_1 \vee C_2$ and let $P_1, \ldots, P_k$ be a maximal system of Menger paths. Put $D_i = P_i \cap D$, $C_{1,i} = P_i \cap C_1$, and $C_{2,i} = P_i \cap C_2$. We have $|C_{1,i}| = 1$ and $|C_{2,i}| \geq 1$ (since every cut must meet each Menger path). We show that $|D_i| \leq |C_{2,i}|$ (then $|D| \leq |D_1| + |D_2| + \cdots + |D_k| + |C_2 \setminus (P_1 \cup \cdots \cup P_k)| \leq |C_2|$). If $C_{1,i} \cap C_{2,i} \neq \emptyset$ then $C_{1,i} \subseteq C_{2,i}$ and we are done, so let $C_{1,i} \cap C_{2,i} = \emptyset$ and let u be the first vertex of $C_{1,i} \cup C_{2,i}$ (in the ordering of $V(P_i)$). It suffices to show that $u \notin D_i$ (then $|D_i| \leq |C_{1,i}| + |C_{2,i}| - 1 \leq |C_{2,i}|$). We have $u \in C_1 \setminus C_2$ or $u \in C_2 \setminus C_1$. In the first case the part of P_i from a to u is not intersected by C_2, so $u \in \alpha(C_2) \subseteq \alpha(D)$, $u \notin D$ and therefore $u \notin D_i$; the second case is similar.

(vi) By (v), $M(G,a,b)$ and $M_0(G,a,b)$ are closed on the operations $\vee$ and $\wedge$. $C_1 \vee C_2$ is the least upper bound for C_1 and C_2, since $\alpha(C_1) \cup \alpha(C_2) = \alpha(C_1 \vee C_2)$ is the least upper bound for $\alpha(C_1)$ and $\alpha(C_2)$ in the ordering of subsets of $V(G)$ by inclusion. By (iv) one may use the duality for showing that $C_1 \wedge C_2$ is the greatest lower bound of C_1 and C_2. □

The previous lemma allows us to treat $M_0(G,a,b)$ as a lattice (with the above-defined ordering $\leq$) and thus speak about maximal (minimal) elements of subsets of $M_0(G,a,b)$.

In the following, let $z: V(G) \to [0,1]$ be a real-valued function on $V(G)$. For a subgraph H of G we shall denote by $z(H)$ the sum of $z(v)$ for all $v \in V(H)$. Usually we shall have $z(G) = 1$, but we permit also $z(G) < 1$ for technical reasons. In algorithms, we shall always assume that z is given by a table and given $v \in V(G)$, the value of $z(v)$ can be found in constant time.

2.5. DEFINITION. Let $\varepsilon \in (0,1)$ be a real number. We call a set $C \subseteq V(G)$ an *ε-splitting* (relative to z) if for every component K of $G \setminus C$ the inequality $z(K) \leq (1-\varepsilon)$ holds.

2.6. LEMMA. *Let* $A, B \in M_0(G,a,b)$, $A < B$, $z(\alpha(A)) \leq \varepsilon$, $z(\beta(B)) \leq \varepsilon$ *and assume that there exists an* $(\varepsilon+\delta)$*-splitting* $C \in R(G,a,b)$. *Then there exists a* δ*-splitting* $D \in R(G,a,b)$ *with* $|D| \leq |C|$ *and* $D \cap \alpha(A) = D \cap \beta(B) = \emptyset$.

Proof. We define $D = (A \vee C) \wedge B$. Since $A < B$, it follows that $A \neq \{b\}$ and $B \neq \{a\}$, and from this we see that $D \in R(G,a,b)$. By Lemma 2.4(v) and its dual we have $|D| \leq |C|$. By definition of the operation $\wedge$, $\beta(B) \subseteq \beta(D)$ and so $\beta(B) \cap D = \emptyset$. Similarly, $\alpha(A) \cap (A \vee C) = \emptyset$ and, since $\alpha(A) \cap B = \emptyset$, it follows that $D \cap \alpha(A) \subseteq ((A \vee C) \cup B) \cap \alpha(A) = \emptyset$.

Let $K_1 = \alpha(D)$, $K_2 = \beta(D)$, $K_3, \ldots, K_m$ be all components of $G \setminus D$. We have $\beta(D) = \beta(B) \cup \beta(A \vee C) \subseteq \beta(B) \cup \beta(C)$ (by Lemma 2.4(ii)) and so $z(\beta(D)) \leq z(\beta(C)) + z(\beta(B)) \leq 1 - \delta - \varepsilon + \varepsilon = 1 - \delta$. Further, $\alpha(D) \subseteq \alpha(A \vee C) = \alpha(A) \cup \alpha(C)$ (apply Lemma 2.4(i) and the dual of Lemma 2.4(ii)) and so $z(\alpha(D)) \leq 1 - \delta$. Finally, $C \subseteq \alpha(A) \cup (A \vee C)$, $A \vee C \subseteq \beta(D) \cup D$, $\alpha(A) \subseteq \alpha(D)$, and so $C \subseteq \alpha(D) \cup D \cup \beta(D)$, and hence for $i > 2$, $C \cap K_i = \emptyset$. Each K_i is contained in some component of $G \setminus C$. This gives $z(K_i) \leq 1 - \delta - \varepsilon < 1 - \delta$, and thus D is a δ-splitting. □

2.7. Lemma. *Let $P_1, \ldots, P_m$ be a system of Menger paths in G between a and b. There is an algorithm which either finds out that $k(G, a, b) > m$ in time $O(|E(G)|)$ or finds the (unique) minimal element C of $M(G, a, b)$ in time $O(|E(\alpha(C) \cup C)|)$.*

Proof. We use one step of the Ford–Fulkerson algorithm. When we reformulate it in terms of undirected graphs and Menger paths instead of networks and flows, we get the following: Let $S \subseteq V(G)$ be defined inductively:

(i) $a \in S$,

(ii) if $u \in S$, and $u = a$ or $u \in V(G) \setminus (V(P_1) \cup \cdots \cup V(P_k))$, then also $v \in S$ for each $\{u, v\} \in E(G)$,

(iii) if $u \in S$, $u \in P_i$, and v precedes u on P_i, then also $v \in S$ and $w \in S$ for each $\{v, w\} \in E(G)$.

If $b \in S$, then $k(G, a, b) > m$ (one can construct a system of $m + 1$ Menger paths); otherwise put $C = \{x_1, \ldots, x_k\}$, where x_i is the maximal vertex in $S \cap V(P_i)$ (in the ordering of vertices of P_i). Then it is not difficult to prove that $C \in M(G, a, b)$, $\alpha(C) = S \setminus C$ and that there is no $C' \in M(G, a, b)$ with $C' < C$. Hence C is as desired. The above description shows that S can be searched in time proportional to $|E(S)|$. □

In the subsequent algorithms, we shall often reduce a graph G by contracting some connected subgraph H of G (containing the vertex a) to a (let us denote the resulting graph by G' for a while). Suppose that $H \subseteq \alpha(C)$ for some $C \in R(G, a, b)$. Then also $C \in R(G', a, b)$, $\alpha(C)$ in G' arises from $\alpha(C)$ in G by the contraction of H, and if $C \in M(G, a, b)$, then also $C \in M(G', a, b)$. Similarly, a system of Menger paths in G is converted to a system of Menger paths in G'. If a function $z\colon V(G) \to [0, 1]$ is given, we define a function z' on $V(G')$ by $z'(a) = z(H)$, $z'(v) = z(v)$ for $v \notin V(H)$. If C is an ε-splitting in G relative to z and $V(H) \cap C = \emptyset$, then C is an ε-splitting in G' relative to z'.

2.8. Lemma. *Let w be a fixed integer. There exists an algorithm with running time $O(|E(G)|)$ which given G, a, b, z as above, an integer k with $0 \leq k \leq w$ and a real number $\varepsilon > 0$ with $z(b) < z(G) - \varepsilon$ produces either*

(i) a (valid) statement that $k(G, a, b) > k$, or

(ii) a set $A \in M_0(G, a, b)$ such that $z(\alpha(A)) \leq \varepsilon$ and A is maximal subject to this property, and a set $B \in M_0(G, a, b)$ such that $B > A$ and B is minimal subject to this property.

Proof. We may assume that $k(G, a, b) = k$ (this can be checked in time $O(|E(G)|)$) and that we have a maximal system of Menger paths $P_1, \ldots, P_k$. The cuts A, B as in (ii) certainly exist, since $z(\alpha(\{a\})) = z(\emptyset) = 0 \leq \varepsilon$ and $z(\alpha(\{b\})) = z(G) - z(b) > \varepsilon$. We proceed as follows:

1. By the previous lemma we find the minimal cut $C \in M(G, a, b)$ and we check if $z(\alpha(C)) \leq \varepsilon$; if not then we return $A = \{a\}$, $B = C$. Otherwise we contract $\alpha(C)$ to a and we modify z and the Menger paths accordingly (we make a work copy of G and z at the beginning, so the original G and z are not destroyed). We mark the vertices of C adjacent to b as "fixed" and the others as "free."

2. We have a current quadruple G, a, b, z, current Menger paths $P_1, \ldots, P_k$, and a current $C \in M(G, a, b)$ which is just the neighborhood of a. Each vertex of C is marked as either "free" or "fixed" in such a way that if $v \in C$ is "fixed," there is no $D \in M_0(G, a, b)$ with $D \geq C$, $v \in D$ and $z(\alpha(D)) \leq \varepsilon$.

If there is no "free" vertex in C, we continue by Step 3. Otherwise we take some "free" $v \in C$ and modify the graph G and the Menger paths by contracting $\{a, v\}$ to a. We apply the procedure of Lemma 2.7 to G, a, b with $m = k$, obtaining either the minimal element D of $M(G, a, b)$ or the answer (i). If the latter case occurs or if $z(\alpha(D)) > \varepsilon$, we restore the changes made on G and on the Menger paths by the contraction of $\{v, a\}$ to a, we mark the vertex v as "fixed" and repeat Step 2. If $z(\alpha(D)) \leq \varepsilon$, we replace the current C by D (marking the vertices of $D \setminus C$: those adjacent to b as "fixed" and the remaining ones as "free"), we contract $\alpha(D)$ to a with the corresponding changes on the Menger paths and z and we repeat Step 2.

3. We put $A = C$ and we find B as any minimal element of the set $\{B_v\colon v \in A\} \subseteq M_0(G, a, b)$, where B_v is the (unique) minimal element of the set $\{D \in M_0(G, a, b)\colon D \geq A \text{ and } v \notin D\}$. Each B_v can be determined in linear time by searching for the minimal element of $M(G_v, a, b)$, where G_v arises from G by contracting $\alpha(A) \cup \{v\}$ to a (it may also happen that v is adjacent to b or that $k(G_v, a, b) > k$; then $B_v = \{b\}$).

Steps 1 and 3 are executed in linear time. Step 2 is repeated at most $|V(G)|$ times.

We observe that a vertex $v \in C$ which is "fixed" will never be removed from the current C, and hence there are at most k repetitions of Step 2, where a vertex becomes "fixed." In other repetitions of Step 2 the time of searching for D is proportional to $|E(\alpha(D) \cup D)|$, and the size of $E(G)$ decreases by the contraction of $\alpha(D)$ at least by $|E(\alpha(D) \cup D)| - w$. Therefore the total execution time is $O(|E(G)|)$. If $K(G, a, b) = k$, the algorithm terminates only with A having all elements "fixed," and hence

A is maximal with respect to $z(\alpha(A)) \le \varepsilon$. If $D \in M_0(G, a, b)$ and $D > A$, then there is some $v \in A \setminus D$ and then $B_v \le D$, thus B is minimal in $M_0(G, a, b)$ with the property $B > A$. □

When we shall choose random vertices of a graph G with a given function $z: V(G) \to [0, 1]$ ($z(G) > 0$), each vertex v of $V(G)$ will have the probability of choice equal to $z(v)/z(G)$ and the choices will be independent.

2.9. Theorem. *Let $w \ge 3$ be a fixed integer. For every δ with $0 < \delta < 1$ there exist $\varepsilon = \varepsilon(\delta) > 0$ and a probabilistic algorithm, which for a given graph G, a function $z: V(G) \to [0, 1]$ with $z(G) = 1$ and a number $p > 0$ yields one of the following kinds of information in time $O(|E(G)|\,|\log p|)$:*

(i) *An ε-splitting $C \subseteq V(G)$ with $|C| \le w$.*

(ii) *A (possibly invalid) statement that there is no δ-splitting $C \subseteq V(G)$ with $|C| \le w$.*

For each G, z such that there is some δ-splitting $C \subseteq V(G)$ of size at most w, the probability that (ii) is (incorrectly) returned is at most p and the result (i) is obtained in expected time $O(|E(G)|)$.

Proof. We fix $\varepsilon = \varepsilon(\delta) = \delta/(w^2 + 1)$. We shall describe a recursive procedure $P(G, z, t)$. Its parameters are:

—a graph G,

—a function $z: V(G) \to [0, 1]$ with $1 - \varepsilon < z(G) \le 1$ and $z(C) \le \varepsilon$ for every $C \subseteq V(G)$, $|C| \le w$, and

—an integer t with $0 \le t \le w$.

The procedure $P(G, z, t)$ returns either an ε-splitting of G of size at most t, or an answer "NO." The description of $P(G, z, t)$ is

1. Let $K_1, \ldots, K_m$ be all components of G. If $z(K_i) \le 1 - \varepsilon$ for each i; then we return $C = \emptyset$. Otherwise for $t = 0$ we return "NO"; for $t > 0$ let $G_1 = K_i$ for the (unique) K_i with $z(K_i) > 1 - \varepsilon$ and let z_1 be z restricted to G_1.

2. We choose vertices $a, b \in V(G_1)$ at random. If $a = b$ or $\{a, b\} \in E(G_1)$ then we return "NO"; otherwise we put $k = 1$ and go to Step 3.

3. If $k > t$, we return "NO." If $z(b) \ge z(G_k) - \varepsilon$ we also return "NO." Otherwise we apply the procedure of Lemma 2.8 to k, $G = G_k$, a, b, and ε. If the answer (i) is returned, we put $G_{k+1} = G_k$, $z_{k+1} = z_k$, we replace k by $k + 1$, and repeat Step 3. Otherwise A, B as in 2.8(ii) were returned. For $B \ne \{b\}$ we check if B is an ε-splitting in G_k and if yes, we return the answer $C = B$; otherwise we go to Step 4.

4. We do Step 5 for vertices $v \in A \cap B \setminus \{a, b\}$ until an ε-splitting is found or all vertices v are exhausted. In the latter case we go to Step 6.

5. Let $G' = G \setminus \{v\}$ and let z' be z restricted to $V(G')$. We call $P(G', z', t - 1)$. If some ε-splitting C' of G' of size $\le t - 1$ is found, we return $C = C' \cup \{v\}$, which is an ε-splitting of G of size $\le t$. Otherwise if "NO" was returned, we continue with the next vertex v.

6. If $A \cap B \ne \emptyset$, or if A and B are adjacent in G_k then we return "NO." Otherwise let G_{k+1} arise from G_k by contracting $A \cup \alpha(A)$ to a and $B \cup \beta(B)$ to b and let z_{k+1} be the accordingly modified function z_k. We replace k by $k + 1$ and we go to Step 3.

This completes the description of $P(G, z, t)$. In each call of $P(G, z, t)$, Steps 1 and 2 are executed at most once, Steps 3 and 6 at most $t \le w$ times, and the loop in Steps 4, 5 is executed at most $t \cdot 2t \le 2w^2$ times (since $|A|, |B| \le k \le t$ in Step 4). Each execution of $P(G, z, t)$ invokes $P(G', z', t - 1)$ at most $2w^2$ times and this recursion has depth at most w. Each elementary step in the procedure P takes a linear time in $|E(G)|$ and the size of G never increases, so the procedure terminates in time $O(|E(G)|)$ (a more careful implementation might considerably reduce the constant of proportionality as well as yield a larger ε for a given δ, but we were not able to remove the exponential dependency of the constant of proportionality on w).

It is easy to check that if the procedure $P(G, z, t)$ returns some C, then it is really an ε-splitting of G of size at most t. Now we shall show by induction on t that the following statement holds for $t = 0, 1, \ldots, w$:

If there is a $(wt + 1)\varepsilon$-splitting C of G of size at most t, then $P(G, z, t)$ returns an ε-splitting of G with probability at least $p_t = \varepsilon^t(1 - 2\varepsilon)^t$. (∗)

For if $\emptyset$ is an ε-splitting, then this is found out in Step 1, and hence the statement holds for $t = 0$. In the following let $t > 0$ and we shall assume that $\emptyset$ is not an ε-splitting (so we obtain a connected graph G_1 with $z(G_1) > 1 - \varepsilon$) and, further, that $C \subseteq V(G_1)$ is a $(wt + 1)\varepsilon$-splitting of G_1 of size $\le t$.

Let $K_1, \ldots, K_m$ be all the components of $G_1 \setminus C$ numbered in such a way that $z(K_1) \ge z(K_2) \ge \cdots \ge z(K_m)$. Since C is a $(wt + 1)\varepsilon$-splitting, we have $z(K_i) \le 1 - (wt + 1)\varepsilon$. By the assumption about the parameters of the procedure $P(G, z, t)$, we have $z(C) \le \varepsilon$ and so $z(G_1 \setminus C) \ge 1 - 2\varepsilon > 1 - (wt + 1)\varepsilon + \varepsilon$; therefore $m \ge 2$ and $z(K_2) + \cdots + z(K_m) \ge \varepsilon$. Thus the probability that a randomly chosen vertex a does not belong to C is at least $1 - 2\varepsilon$ and, if this happens, the probability that a next randomly chosen vertex b lies in another component of $G_1 \setminus C$ than a is at least ε. Therefore the probability that $C \in R(G_1, a, b)$ is at least $(1 - 2\varepsilon)\varepsilon$.

Now we shall prove the following statement by backward induction on k:

Suppose that the execution of Step 3 starts for some k, G_k, a, b such that $k \leq t$, $k(G_k, a, b) \geq k$ and there exists a $(wt - k + 2)\varepsilon$-splitting $C_k \in R(G_k, a, b)$ with $|C_k| \leq t$. Then the probability that the procedure $P(G, z, t)$ terminates by returning an ε-splitting is at least p_{t-1}. $(**)$

If the statement $(**)$ holds for some t and $k = 1$, then this establishes the validity of $(*)$ for this t, since after execution of Step 2 the hypotheses of $(**)$ are satisfied with probability at least $(1 - 2\varepsilon)\varepsilon$.

Suppose that $(*)$ holds for $t < s$ $(s \geq 1)$ and that $(**)$ holds for $t = s$ and $k = m + 1, m + 2, \ldots, s$ (the second hypothesis is void for $m = s$). Further suppose that the hypotheses of $(**)$ are satisfied for $t = s$ and $k = m$. We shall show that $(**)$ holds also for $t = s$ and $k = m$.

Among others, the hypotheses of $(**)$ for $t = s$ and $k = m$ imply that $z(b) \leq 1 - (ws + 1)\varepsilon < 1 - 2\varepsilon \leq z(G_m) - \varepsilon$. Since $C_m \in R(G_m, a, b)$ and $|C_m| \leq s$, we have $k(G_m, a, b) \leq s$. If $k(G_m, a, b) > m$, then Step 3 is immediately repeated with $k + 1$ instead of k, and hence $(**)$ for $t = s$, $k = m$ follows from the validity of $(**)$ for $t = s$, $k = m + 1$.

Now suppose that $k(G_m, a, b) = m$. Then the cuts $A, B \in M(G_m, a, b)$ as in 2.8(ii) are found in Step 3. Since A is maximal with respect to the property $z(\alpha(A)) \leq \varepsilon$, it follows that $z(\alpha(B)) > \varepsilon$. If also $z(\beta(B)) > \varepsilon$, then B would be an ε-splitting of G_m (and also of G) of size $m \leq t$ and it would be returned to Step 3. Assume that $z(\beta(B)) \leq \varepsilon$. Then the assumptions of Lemma 2.6 are satisfied for $A, B, C = C_m$, $\delta = (ws - m + 1)\varepsilon$, and so there exists a $(ws - m + 1)\varepsilon$-splitting $C_{m+1} \in R(G_m, a, b)$ with $|C_{m+1}| \leq s$, $\alpha(A) \cap C_{m+1} = \beta(B) \cap C_{m+1} = \varnothing$. Let us distinguish two cases:

(i) $C_{m+1} \cap (A \cup B) \neq \varnothing$. Let $v \in C_{m+1} \cap (A \cup B)$. Then $C_{m+1} \setminus \{v\}$ is a $(ws - (m + 1) + 2)\varepsilon$-splitting in $G \setminus \{v\}$ of size $\leq s - 1$ and by $(*)$ for $t = s - 1$ (note that $ws - (m + 1) + 2 \geq w(s - 1) + 1$) a $(w(s - 1) + 1)\varepsilon$-splitting is found in Steps 4 and 5 with probability at least p_{s-1}, so in this case $(**)$ holds for $t = s$, $k = m$.

(ii) $C_{m+1} \cap (A \cup B) = \varnothing$. First we show that this cannot happen for $m = s$: if it did, then C_{m+1} would be an element of $M(G_m, a, b)$ (since $k(G_m, a, b) = m = s$) with $A < C_{m+1} < B$, which is impossible since B was minimal with $B > A$.

Now let $m > s$. We have $C_{m+1} \cap (A \cup B \cup \alpha(A) \cup \beta(B)) = \varnothing$, thus C_{m+1} is also an element of $R(G_{m+1}, a, b)$ and a $(ws - (m + 1) + 3)\varepsilon$-splitting of G_{m+1} of size $\leq s$. Further, $k(G_{m+1}, a, b) \geq m + 1$, since if it were $k(G_{m+1}, a, b) = m$, we would get a cut $D \in M(G_{m+1}, a, b)$ of size m and this would be also an element of $M(G_m, a, b)$ with $A < D < B$, which is impossible. Thus the assumptions of $(**)$ are satisfied for $t = s$ and $k = m + 1$ and this implies $(**)$ also for $t = s$, $k = m$.

Now the algorithm for Theorem 2.9 will be the following: First we check if there is $C \subseteq V(G)$ with $|C| \leq w$ and $z(C) \geq \varepsilon$ (this can be done in linear time by finding the vertices with w largest values of z) and if yes, we return C as an answer (it is certainly an ε-splitting). In the opposite case the triple (G, z, w) satisfies the assumptions on parameters of the procedure $P(G, z, t)$. We shall repeatedly call the procedure $P(G, z, w)$ until an ε-splitting is found (and in this case we immediately return it) or until the number of calls reaches $Q = \lceil |\log p| / |\log(1 - p_w)| \rceil$ (and in this case we return the statement (ii)). Let $\delta > 0$ be a real number and let $\varepsilon = \varepsilon(\delta)$. If a δ-splitting of size $\leq w$ exists in G, then in each call of $P(G, z, w)$ the probability that it does not find an ε-splitting is (by $(*)$) at most $1 - p_w$ and the results of different calls are independent, so the probability that an ε-splitting is not found in Q calls is at most $(1 - p_w)^Q \leq p$. The expected number of calls needed for finding an ε-splitting is estimated by $p_w + 2p_w(1 - p_w) + 3p_w(1 - p_w)^2 + \cdots$, which is finite and depends only on p_w; thus the expected running time for obtaining a positive solution is $O(|E(G)|)$. □

3. Finding A Tree-Decomposition

In this section we prove Theorem 1.1. If $Z \subseteq V(G)$ is a nonempty set, we assign to it the *characteristic function* z_Z: $z_Z(v) = 1/|Z|$ for $v \in Z$ and $z_Z(v) = 0$ otherwise. Let $w \geq 3$ be a fixed integer. In view of Theorem 1.4 it suffices to prove Theorem 1.1 for $w \geq 3$.

3.1. Lemma. *Let G be a graph of tree-width $< w$ and let $Z \subseteq V(G)$, $Z \neq \varnothing$. Then there exists a $(1/3)$-splitting $C \subseteq V(G)$ of size at most w (with respect to the characteristic function of Z).*

Proof. This follows from (2.6) of [6]. □

Let $Z \subseteq V(G)$. A tree-decomposition (T, τ) of G will be called a *Z-decomposition* if it has width $\leq 6w$ and there exists a vertex $t \in T$, called a *Z-vertex*, such that $Z \subseteq \tau(t)$.

We shall describe two recursive procedures $A(G, Z, q)$ and $B(G, Z, q)$ which will be used for the algorithm of Theorem 1.1. The parameters of the procedures are: a graph G, a set $Z \subseteq V(G)$ with $|Z| \le 6w$ (with $|Z| \le 5w$ for $B(G,Z,q)$) and a real number $q > 0$. Both procedures either return a Z-decomposition of G of width $\le 6w$ or an answer "NO," which should be interpreted as a statement "the tree-width of G is $\ge w$." This last statement may be false, but with probability bounded below 1 by some function of q and $|V(G)|$. The description of the procedures contains a constant N_0 (depending on w only) which will be determined later. The description of $A(G,Z,q)$ is

1. We check whether $|E(G)| \le w|V(G)|$; if not we return "NO." This answer is correct by 1.5(iii).

2. If $|V(G)| \le N_0$, we proceed by brute force (examining all possible decompositions). This takes only a constant time and the answer is certainly correct. For $|V(G)| > N_0$ we go to the next step.

3. We find a (1/3)-splitting $C \subseteq V(G)$ relative to z_Z with $|C| \le w$ (for $Z = \emptyset$ we choose $C = \emptyset$). This can be done in time $O(|E(G)|)$ by checking all partitions of Z into Z_1, Z_2, and Z_3 with $|Z_1|, |Z_2| \le (2/3)|Z|, |Z_3| \le w$; for each such partition we compute if there is a cutset of size $\le w - |Z_3|$ separating Z_1 from Z_2 in $G \setminus Z_3$ (for details see also [11]). If such a (1/3)-splitting does not exist, we return "NO" (this is a correct answer by Lemma 3.1). Otherwise we can produce a separation (G_1, G_2) of G such that $|V(G_1) \cap V(G_2)| \le w$ and $|(V(G_1) \setminus V(G_2)) \cap Z| \le 4w$, $|(V(G_2) \setminus V(G_1)) \cap Z| \le 4w$. Let $Z_i = (V(G_1) \cap V(G_2)) \cup (V(G_i) \cap Z)$, $i = 1,2$. Clearly $|Z_i| \le 5w$. We call $B(G_1, Z_1, q)$ and $B(G_2, Z_2, q)$.

4. If one of the above calls yields answer "NO," we return "NO." If one these "NO" answers was correct, our "NO" answer is also correct by Proposition 1.5(i). If neither of the calls gives "NO," we have a Z_i-decomposition (T_i, τ_i) of G_i with Z_i-vertex t_i. Let $t_0 \notin V(T_1) \cup V(T_2)$ be a new vertex. We define a pair (T, τ) by $V(T) = V(T_1) \cup V(T_2) \cup \{t_0\}$, $E(T) = E(T_1) \cup E(T_2) \cup \{\{t_0, t_1\}, \{t_0, t_2\}\}$, $\tau(t) = \tau_1(t)$ for $t \in V(T_1)$, $\tau(t) = \tau_2(t)$ for $t \in V(T_2)$, and $\tau(t_0) = Z$. Then (T, τ) is a Z-decomposition of G and we return it as an answer.

The description of $B(G,Z,q)$: Steps 1, 2, and 4 are the same as for $A(G,Z,q)$. Step 3 is replaced by

3′. We apply the probabilistic algorithm of Theorem 2.9 for $\delta = \frac{1}{3}$ to the graph G, function $z = z_{V(G)}$, and probability $p = q$. If the answer (ii) is obtained, we return "NO" (which may be incorrect). Otherwise we are returned some ε-splitting $C \subseteq V(G)$ with $|C| \le w$ (where $\varepsilon = \varepsilon(1/3)$ is as in Theorem 2.9). We can produce a separation (G_1, G_2) of G such that $|V(G_i)| \le (1-\varepsilon)|V(G)|$ $(i = 1,2)$ and $|V(G_1) \cap V(G_2)| \le w$. We put $Z_i = (V(G_1) \cap V(G_2)) \cup (Z \cap V(G_i))$, $i = 1,2$. Since we assume that $|Z| \le 5w$, we have $|Z_i| \le 6w$. We call $A(G_1, Z_1, q)$ and $A(G_2; Z_2, q)$.

To obtain our algorithm for Theorem 1.1, we call $A(G, \emptyset, p^2/|V(G)|^2)$. To estimate the running time of this algorithm, let $a(n,q)$ ($b(n,q)$, respectively) be the worst-case running time of $A(G,Z,q)$ ($B(G,Z,q)$, respectively) for $|V(G)| \le n$. Steps 1–4 are executed in linear time, Step 3′ is executed in time $O(|E(G)| \cdot |\log q|)$. This yields recurrent formulae

$$a(n,q) \le \text{const} \quad \text{for } n \le N_0,$$
$$a(n,q) \le O(n) + \max\{b(n_1 + w, q) + b(n_2 + w, q); n_1 + n_2 \le n\},$$
$$b(n,q) \le \text{const} \quad \text{for } n \le N_0,$$
$$b(n,q) \le O(n|\log q|) + \max\{a(n_1 + w, q) + a(n_2 + w, q); \; n_1 + n_2 \le n, n_i \le (1-\varepsilon)n\}.$$

If we choose $N_0 > 4w/\varepsilon$ (another requirement on N_0 is made later), then $(1-\varepsilon)n + 2w \le (1 - \varepsilon/2)n$ for $n \ge N_0$ and one can verify that then $a(n,q) = O(n \log n |\log q|)$, and hence the total running time of our algorithm is $O(n(\log n)(\log n + |\log p|))$. The expected time for successful completion of the algorithm is derived similarly.

The algorithm can produce an incorrect answer only at Step 3′. Obviously Step 3′ can be executed at most $a(n,q)$ times, and the probability that it gives an incorrect answer at least once is thus at most

$$q \cdot a(q,n) \le (p^2/n^2) \cdot \text{const} \cdot n \log^2 n |\log p| \le p$$

for $n \ge N_0$, if N_0 is sufficiently large. The correctness of the algorithm follows from the description.

4. The Case $w \le 3$

In this section we prove Theorem 1.4.

4.1. Definition. A *reduction* R is a pair (H, d), where H is a graph and d: $V(H) \to \omega + 1$ is a labeling of vertices of H by ordinals (finite ones and ω), such that there exists some $v \in V(H)$ with $d(v) = 0$. We say that $v \in V(H)$ is an *inner vertex* of R if $d(v) = 0$, otherwise v is an *outer vertex*. If $d(v) = \omega$ then v is an *unbounded vertex*, otherwise v is a *bounded vertex*. We say that R *occurs in* G and that ψ is an *occurrence* of **R in G**, if ψ is an injective mapping from $V(H)$ into $V(G)$ preserving **edges** (that is, if u, v are adjacent in H, then $\psi(u), \psi(v)$ are adjacent in G) **and such** that for every $v \in V(H)$ the number of edges joining $\psi(v)$ to **$V(G) \setminus \text{Im}\,\psi$** is at most $d(v)$. In this case we define a new graph–the result of application of R on G—which arises from G by deleting all ψ-images of inner vertices of R, by adding a complete subgraph on the set of ψ-images of outer vertices of R and by deleting multiple edges produced by this. If ψ is an occurrence of a reduction (H, d) and $v \in V(H)$ is an inner (outer, bounded, unbounded) vertex of R, then we also call $\psi(v)$ an inner (outer, bounded, unbounded) vertex of the occurrence ψ.

We shall represent reductions by pictures of their underlying graphs H with vertices labeled by the values of d according to the following conventions: the bounded vertices are represented by small black circles, the unbounded vertices by small empty circles, and the labels 0 and ω are omitted.

4.2. Definition. We say that a set S of reductions is *safe* for a class $\mathscr{F}$ of graphs, if an application of a reduction from S to a graph preserves both its membership and nonmembership in $\mathscr{F}$. A set S of reductions is *complete* for $\mathscr{F}$ if for every non-null graph $G \in \mathscr{F}$ there is a reduction $R \in S$ occurring in G.

Arnborg and Proskurowski [3] proved that the set of reductions (a), ..., (f) of Fig. 1 is safe and complete for the class of graphs of

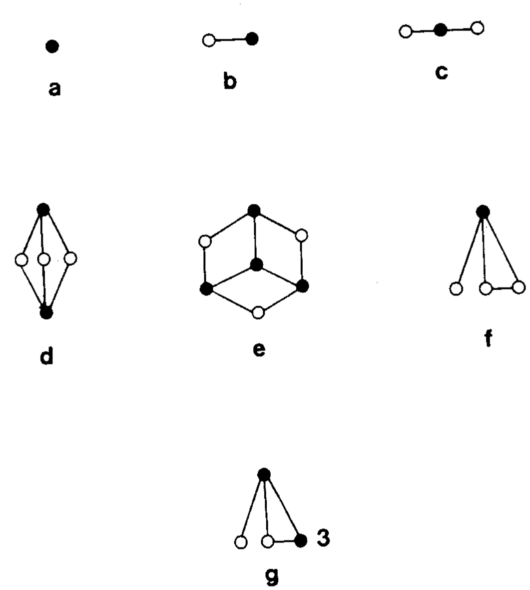

FIG. 1

tree-width ≤ 3 and that the set (a), (b), (c) is safe and complete for the class of graphs of tree-width ≤ 2. This implies that a graph G has tree-width ≤ 3 if and only if it can be reduced to the null graph by repeated application of these reductions. Moreover, it is easy to construct a tree-decomposition of G of width ≤ 3 from the recorded sequence of these reductions. We need to strengthen the result of Arnborg and Proskurowski as follows.

4.3. Lemma. *The set of reductions* (a), ..., (e) *and* (g) *of Fig.* 1 *is safe and complete for the class of graphs of tree-width* ≤ 3.

Proof. The safeness follows from 1.5(i), because if H is the graph obtained from G by applying one of the reductions (a), ..., (e) or (g), then H is isomorphic to a minor of G.

To prove that the set is complete let us suppose for a contradiction that G is a non-null graph of tree-width ≤ 3 such that none of the reductions (a), ..., (e), (g) occurs in G. Then each vertex of G has degree at least 3. We may assume that G is connected.

By 1.1(ii) there exists a chordal graph H containing no K_5 with $V(H) = V(G)$ and $E(G) \subseteq E(H)$; we deduce that H is not a complete graph. We say that a vertex $v \in V(H)$ is a *1-leaf* if its neighbors form a complete subgraph in H. We say that $v \in V(H)$ is a *2-leaf* if it is is a 1-leaf in the graph arising from H by deleting all 1-leaves. By a well-known property of chordal graphs, each chordal graph contains at least one 1-leaf and this implies that each connected chordal graph which is not a complete graph contains at least one 2-leaf.

Let u be a 2-leaf of H and let $v_1, \ldots, v_m$ be all the 1-leaves adjacent to it. Denote $X = \{u\} \cup \{v\colon \{u, v\} \in E(H)\} \setminus \{v_1, \ldots, v_m\}$. By the definition of a 2-leaf, we get that the subgraph induced by X is complete and so $|X| \leq 4$. The degree of each v_i in G is 3 (if it were 4 we would get K_5 in H); we denote by N_i the set of neighbors of v_i in G. We have $N_i \subseteq X$ by the definition of 1-leaf. Necessarily $m \leq 3$, since otherwise the reduction (d) would have to occur in G. This implies that the degree of u in G is at most $|X| - 1 + m \leq 6$.

We shall consider v_1 and its neighbors, let us call them u, u_1, and u_2. Then if $\{u, u_1\} \in E(G)$ or $\{u, u_2\} \in E(G)$, we have the occurrence of (g) (see Fig. 2), and so we assume the contrary. But u has degree at least 3, and therefore $m \geq 2$. Consider v_2 and its neighbors. These cannot be $\{u, u_1, u_2\}$ (since we would have an occurrence of (d)); let it be u, u_1, u_3 (then $X = \{u, u_1, u_2, u_3\}$). As before we assume $\{u, u_3\} \notin E(G)$ (otherwise (g) occurs), so u having degree at least 3 forces $m \geq 3$. Now u_1 cannot be a neighbor of v_3 (reduction (d) would occur) and so its neighbors are just u, u_2, u_3. Now $v_1, v_2, v_3, u, u_1, u_2, u_3$ induce an occurrence of (e), a contradiction. □

Our algorithm for recognition of graphs of tree-width ≤ 3 will proceed by applying the reductions from Lemma 4.3 (in the sequel, by a reduction we shall mean a reduction from this set) on a current graph, yielding a (smaller) modified graph, until either the graph becomes null, or none of the reductions is applicable.

The graph in our algorithm will be represented as follows: Vertices are referred to by numbers $1, 2, \ldots, n$, and for each vertex we shall maintain a doubly linked list of edges incident to it (in which an edge may appear several times). The entry for each edge contains the number of the other end of that edge and a pointer to the entry for the same edge in the other end's edge list.

We need two technical lemmas.

4.4. LEMMA. *Let G be a graph and let $U \subseteq V(G)$ be a set of some vertices of degree 3 in G. We call $u, v \in U$ equivalent if they have the same set of neighbors in G. The classes of this equivalence on U can be found in time $O(|U|)$ (provided that we have a list of members of U).*

Proof. We form the set V of all neighbors of vertices from U ($|V| \leq 3|U|$) and a graph $H = (V \cup U, \{\{u, v\}: u \in U, v \in V, \{u, v\} \in E(G)\})$, having at most $3|U|$ edges. Now for each $a \in V$ we find $U_a = \{u \in U: (u, a) \in E(H)\}$ and $V_a = \{v \in V \setminus \{a\}; \{u, v\} \in E(H)$ for some $u \in U_a\}$. We form H_a, which will be the restriction of H to $U_a \cup V_a$. Now similarly for each $b \in V_a$, we make U_{ab} (the set of neighbors of b in H_a), V_{ab} (the set of neighbors of U_{ab} in $H_a \setminus \{b\}$), and the graph H_{ab} (the restriction of H to $U_{ab} \cup V_{ab}$), and finally for each $c \in V_{ab}$ we find U_{abc} (the set of all neighbors of c in H_{ab}). Now the equivalence classes on U are just the sets U_{abc} (and each is obtained exactly once if we take only a, b, c with $a < b < c$ in an arbitrary ordering of V) and the whole procedure takes a constant time per edge of H (since each edge of H occurs in at most two graphs H_a, each edge of H_a occurs in at most one H_{ab} and each edge of H_{ab} is incident to a vertex of at most one set U_{abc}). □

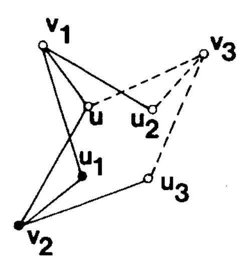

FIG. 2

4.5. LEMMA. *Given a list of k integers in range $[1, n]$, one can detect (and delete) all multiple occurrences of items in this list in time $O(k)$ and space $O(n)$.*

Proof. We shall use a table indexed by $1, 2, \ldots, n$, where the intended meaning of entry i is whether the element i has already been encountered. In the first passage through the list we initialize the entries for the members of the list to "false," and in the second passage we can already detect the multiple occurrences. □

4.6. *Proof of Theorem* 1.4. We prove the theorem for $w = 3$, for the case $w = 2$ can be obtained by a straightforward modification, and the case $w = 1$ is easy.

We wish to proceed by applying the reductions. Every edge of an occurrence of a reduction in a graph G is incident to a vertex of degree ≤ 6 in G, and given a vertex v of G of degree ≤ 6 one can find all occurrences of reductions (a), (b), (c), (e), (g) having v as a bounded vertex in constant time. Occurrences of reduction (d) can be dealt with using Lemma 4.4. Also, insertions of edges and deletions of edges incident to inner vertices are easily handled in constant time, but what is not so obvious is how to delete multiple edges. We proceed as follows. If a vertex has degree at most 6, we delete multiple edges incident to it immediately; otherwise we delay the deletion and process multiple edges in batches.

Let us describe the algorithm now.

Step 1. We check if $|E(G)| > 3|V(G)|$ and if so we answer that the tree-width of G is > 3 and stop. Otherwise, we put $p = 1$, $H = G_1 = G$, $OC(v) = 0$, and $DE(v) = \deg_G(v)$ for every $v \in V(G)$, and let S_1 be the set of all vertices of G_1 of degree at most 6.

Step 2. Let S' be the set of all vertices $v \in S_p$ with $\deg_{G_p}(v) \leq 6$. We find all occurrences of reductions (a), (b), (c), (e), (g) in $H = G$ with a bounded vertex in S', and the classes of vertices of S' o degree 3 having the same neighborhood in H. Thus we represen all occurrences of reductions with a bounded vertex in S'. We g to Step 3.

Step 3. We initialize S_{p+1} to the empty set. For each reduction R founc in Step 2 we check if it is a reduction in H and if so we do th following.

Let $u_1, \ldots, u_m$ be the inner vertices of R and let $v_1, \ldots, v_l$ be th outer vertices of R. For every edge $e = \{u_i, v\}$ incident with som u_i ($i = 1, \ldots, m$) we delete e from H, and call Step 5 wit $w_1 = u_i, w_2 = v$. For every pair v_i, v_j ($i, j = 1, \ldots, l, i \neq j$) of oute vertices we add an edge joining v_i, v_j to (the edge-list of) H an call Step 5 with $w_1 = v_i, w_2 = v_j$.

After exhausting all reductions we go to Step 4.

Step 4. For each equivalence class E found in Step 2 we do the following. Let $N = \{u_1, u_2, u_3\}$ be the common neighborhood of vertices of E in G_p. We first discard all vertices whose neighborhood in H is not N. Let $\{v_1, \dots, v_l\}$ be the remaining vertices of E. If $l \le 1$ we go to the next equivalence class. Otherwise for every $i = 1, \dots, l$ and every $j = 1, 2, 3$ we delete the edge $\{v_i, u_j\}$ from H and call Step 5 with $w_1 = v_i, w_2 = u_j$, and for every $i, j = 1, 2, 3$ with $i \ne j$ we add an edge joining u_i, u_j to (the edge-list of) H and call Step 5 with $w_1 = u_i, w_2 = u_j$.
After exhausting all equivalence classes we go to Step 7.

Step 5. We place vertices w_1, w_2 on a stack and repeat Step 6 until the stack becomes empty. Then we return.

Step 6. Pop a vertex w from the stack. Add w to S_{p+1}. Increment $OC(w)$ by one. If $DE(w) \le 12$ or if $OC(w) > DE(w)/2$ remove the multiple edges occurring in the edge-list of w in H, set $OC(w)$ to 0, set $DE(w)$ to $\deg_H(w)$ and push all the other ends of deleted edges on the stack.

Step 7. We put $G_{p+1} = H$. If G_{p+1} is the null graph we answer that the tree-width of G is ≤ 3 and stop. Otherwise we go to Step 8.

Step 8. If $S_{p+1} = \emptyset$ we answer that the tree-width of G is > 3 and stop. Otherwise we replace p by $p + 1$, and go to Step 2 for next iteration.

This completes the description of the algorithm. We claim that after the completion of Step 5 the following conditions are satisfied for every $v \in V(H)$:

(1) $OC(v) \le DE(v)/2$.

(2) $|DE(v) - \deg_H(w)| \le OC(v)$,

(3) *if* $\deg_H(v) \le 6$ *then* $DE(v) = \deg_H(v)$ *and the edge-list of* v *contains no duplicate entries.*

For (1) and (2) follow easily. From (1) and (2) we deduce that $DE(v) \le 2\deg_H(v)$, and hence (3) follows. It follows from (3) that $\deg_H(v) \le 6$ can be decided in constant time.

(4) *If* $\deg_H(v) \le 6$ *at any time during the execution of the algorithm, then* $v \in S_p$, *for some* $p = 1, 2, \dots$.

For either $\deg_G(v) \le 6$ in which case $v \in S_1$, or at some stage of say the pth iteration $\deg_H(v) = 7$ and we delete an edge incident to v. Then v is included in S_{p+1} at Step 6, as desired.

Now we prove the correctness of the algorithm. The answer of Step 1 is correct by Lemma 1.5(iii), the answer of Step 7 is correct by Lemma 4.3. It remains to prove that the answer of Step 8 is correct. So suppose for a contradiction that the algorithm terminated at Step 8 with $p = m$, with $S_{m+1} = \emptyset$, and with G_{m+1} (non-null and) of tree-width ≤ 3. By Lemma 4.3 there exists an occurrence ψ of a reduction in G_{m-1}; let $v_1, \dots, v_n$ be its bounded vertices. Then $v_i \in S_1 \cup S_2 \cup \cdots \cup S_m$ for every $i = 1, \dots, n$ by (4), and let i, j be such that $v_i \in S_j$ and $\{v_1, \dots, v_n\} \cap (S_{j+1} \cup \cdots \cup S_m) = \emptyset$.

The occurrence of ψ did not exist continuously in H since the beginning of the jth iteration, for otherwise it would have been detected and destroyed. Therefore ψ was created (or recreated) at a later step, but at the time it was created, a bounded vertex of ψ was added to $S_{j+1} \cup \cdots \cup S_m$, a contradiction. This proves the correctness of the algorithm.

To estimate the running time of the algorithm we first estimate the running time of each step. Step 1 takes time $O(|V(G)|)$, Step 2 takes time $O(|S_p|)$ by Lemma 4.4, Steps 3 and 4 also take time $O(|S_p|)$, because all the deletions and additions of edges can be done in constant time. Step 5 takes time $O(1)$ each time it is called. Steps 7 and 8 take time $O(1)$. Before we examine Step 6 we need the following.

(5) *Let* I *be the total number of edge insertions and edge deletions performed during the execution of the algorithm. Then* $I = O(|V(G)|)$.

For there are at most three additions and at most 12 deletions per reduction, and the total number of reductions applied is $\le |V(G)|$ because every application of a reduction decreases $|V(H)|$.

Step 6 is called at most $2I$ times, because a vertex is pushed on the stack only after deletion or insertion of an edge. Let $w \in V(H)$. We say that Step 6 is *executed using* w if w is the vertex obtained at the beginning of Step 6. If Step 6 is executed using w, and $DE(w) \le 12$ or $OC(w) \le DE(w)/2$, then the execution time is $O(1)$: otherwise it is $O(DE(w) + OC(w)) = O(OC(w))$ by Lemma 4.5. But in the latter case there are $OC(w) - 1$ previous executions using w in which the execution time was constant. Therefore the total time spent in Step 6 is $O(|V(G)|)$. Since obviously $|S_2| + |S_3| + \cdots \le 2I$, we deduce that the overall running time is $O(|V(G)|)$. □

5. Discussion and Open Problems

The algorithm of Theorem 1.4 is relatively simple and practical, but the algorithm of Theorem 1.1 is not, because the constant of proportionality is enormous even for small values of w. A natural question is, of course, whether there is a deterministic algorithm with the same running time. A related problem is whether there exist a constant $C > 0$ and a polynomial algorithm which given a graph G and an integer $w > 0$ either produces a tree decomposition of G of width $\leq Cw$, or (correctly) answers that the tree-width of G is $\geq w$. Let us remark that such an algorithm exists if we replace Cw by Cw^4. Finally, does there exist a "local characterization" of graphs of tree-width $\leq w$ for $w > 3$, similarly as in Lemma 4.3 for $w = 3$?

ACKNOWLEDGMENT

We are indebted to Hans L. Bodlaender and Eva Tardos for pointing out oversights in an earlier version of this paper. We also thank the referees for helpful comments.

REFERENCES

1. S. ARNBORG, D. G. CORNEIL AND A. PROSKUROWSKI, Complexity of finding embeddings in a k-tree, *SIAM J. Algebraic Discrete Methods* **8** (1987), 277–284.
2. S. ARNBORG AND A. PROSKUROWSKI, Linear time algorithms for NP-hard problems on graphs embedded in k-trees, *Discrete Applied Math.* **23** (1989), 11–24.
3. S. ARNBORG AND A. PROSKUROWSKI, Characterization and recognition of partial 3-trees, *SIAM J. Algebraic Discrete Methods* **7** (1985), 305–314.
4. J. A. WALD AND C. J. COLBOURN, Steiner trees, partial 2-trees and minimum IFI networks, *Networks* **13** (1983), 159–167.
5. J. MATOUŠEK AND R. THOMAS, On the complexity of finding iso- and other morphisms for partial k-trees, submitted.
6. N. ROBERTSON AND P. D. SEYMOUR, Graphs minors. II. Algorithmic aspects of tree-width, *J. Algorithms* **7** (1985), 309–322.
7. N. ROBERTSON AND P. D. SEYMOUR, Graph minors. III. Planar tree-width, *J. Combin. Theory Ser. B* **36** (1984), 49–64.
8. N. ROBERTSON AND P. D. SEYMOUR, Graph minors. IV. Tree-width and well quasi-ordering, *J. Combin. Theory Ser. B* **48** (1990), 227–254.
9. N. ROBERTSON AND P. D. SEYMOUR, Graph minors. V. Excluding a planar graph, *J. Combin. Theory Ser. B* **41** (1985), 92–114.
10. N. ROBERTSON AND P. D. SEYMOUR, Graph minors. X. Obstructions to tree-decompositions, submitted for publication.
11. N. ROBERTSON AND P. D. SEYMOUR, Graph minors. XIII. The disjoint paths problem, submitted for publication.

JOURNAL OF ALGORITHMS **12**, 23–37 (1991)

An Improved Algorithm for the Planar 3-Cut Problem

XIN HE

Department of Computer Science, State University of New York at Buffalo, Buffalo, New York 14260

Received August 1, 1988; revised November 28, 1989

A 3-cut of a connected graph G is a subset of edges which, when deleted, separates G into three connected components. In this paper we present an $O(n \log n)$ algorithm for finding a minimum 3-cut in planar graphs. Our algorithm improves the best previously known algorithm for the problem by an $O(n/\log n)$ factor. © 1991 Academic Press, Inc.

1. INTRODUCTION

Let $G = (V, E)$ be a connected simple graph (i.e., no self-loops and multiple edges) with n vertices. A *k-cut* of G is a subset of edges $E_1 \subseteq E$ such that the graph $G_1 = (V, E - E_1)$ has k connected components. The minimum k-cut problem is to find a k-cut in G with minimum number of edges. This problem is NP-complete for arbitrary k [11]. Recently Goldschmidt and Hochbaum showed that the problem can be solved in $O(n^{k^2})$ time for a fixed k [11].

We consider in this paper a special case for this problem: Finding a minimum 3-cut in planar graphs. This special case is interesting not only as an extension of the ordinary 2-cut problem, but also because of its applications in cutting plane methods for the traveling salesperson problem [4].

There is a trivial 3-cut of size at most 10 in any planar graph G: Let u be a vertex in G with minimum degree and let E_1 be the set of the edges adjacent to u. Since the minimum degree of the vertices of G is at most 5 [2], $|E_1| \leq 5$. Let v be a vertex in the graph $G - \{u\}$ with minimum degree and let E_2 be the set of the edges adjacent to v. As above we have $|E_2| \leq 5$. The union of E_1 and E_2 is a 3-cut with size at most 10. To find a minimum 3-cut we simply check all subsets of edges with size ≤ 10. This

0196-6774/91 $3.00
Copyright © 1991 by Academic Press, Inc.
All rights of reproduction in any form reserved.