FINDING APPROXIMATE SEPARATORS AND COMPUTING TREE WIDTH QUICKLY

Bruce A. Reed *)

December 1991

ABSTRACT

We show that for any fixed k, there is a linear-time algorithm which given a graph G either:

(i) finds a cutset X of G with $|X| \leq k$ such that no component of $G - X$ contains more than $\frac{3}{4}|G - X|$ vertices, or

(ii) determines that for any set X of vertices of G with $|X| \leq k$, there is a component of $G - X$ which contains more than $\frac{2}{3}|G - X|$ vertices.

This approximate separator algorithm can be used to develop an $\mathcal{O}(n \log n)$ algorithm for determining if G has a tree decomposition of width at most k (for fixed k) and finding such a tree decomposition if it exists.

1. Introduction

"Divide and Conquer" and "Dynamic Programming" are two of the most general and effective techniques for designing efficient algorithms. Many classes of graphs can be decomposed in a tree-like fashion, allowing us to solve difficult optimization problems quickly via a combination of these two techniques.

This approach was first applied to trees and chordal graphs [20]. A fundemental breakthrough came with the seminal paper of Robertson and Seymour [30] in which they defined tree decompositions and tree width. We shall define these notions precisely in the next section, for now it will suffice to think of a tree decomposition as a partition of the edges of G into a set of subgraphs which fit together in a tree-like way. The width of a tree decomposition is the number of vertices in the largest of these subgraphs and the tree width of a graph is the minimum of the widths of its tree decompositions (actually Robertson and Seymour subtract 1 from these values, we find it more convenient not to do so). Robertson and Seymour gave a polynomial-time algorithm which given a graph G either determined that G had tree width greater than k or found a tree decomposition of width $4k$.

Subsequently, a host of researchers have written a raft of papers showing that given a

---

*) Forschungsinstitut für Diskrete Mathematik, Universität Bonn, Nassestr. 2, 5300 Bonn 1, Germany. Supported by Sonderforschungsbereich 303 (DFG) and the Alexander von Humboldt-Stiftung.
Permanent address: Dept. of Combinatorics and Optimization, University of Waterloo, Ontario, Canada NZL 361.

24th ANNUAL ACM STOC - 5/92/VICTORIA, B.C., CANADA
© 1992 ACM 0-89791-512-7/92/0004/0221...$1.50

bounded-width decomposition of a graph G, many difficult (often NP-complete) optimization problems can be solved in polynomial time [3,5,8,11,12,13,14,17,18,21,28,34]. Examples include Hamiltonicity, Graph Isomorphism, Vertex Colouring, Edge Colouring and various routing problems. In fact, it has been shown that any problem which can be expressed in certain languages [17,18] or by certain types of logical expressions [3] can be solved in polynomial time on graphs for which we are given a bounded width tree decomposition . Combining these results with Robertson and Seymour's original algorithm we see that there are polynomial-time algorithms which solve all these problems for graphs of bounded tree width.

Most of the problems mentioned above can actually be solved in linear time, once we have the desired tree decomposition . Thus the dominating factor in these algorithms is the time taken to create this tree decomposition . Robertson and Seymour's original algorithm took such a long time they didn't even bother to compute the exponent of the polynomial, they simply stated that it was a function of k. This was improved by Arnborg, Corneil, and Proskurowski [6] who gave an $\mathcal{O}(n^{k+2})$ algorithm and then again by Robertson and Seymour [32] who developed an $\mathcal{O}(n^2)$ algorithm. One of the main applications of the result I discuss in this paper is that it can be used to develop an $\mathcal{O}(n \log n)$ algorithm for this problem thereby improving the running time of all the algorithms mentioned above from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. I note that Bodlaender and Kloks [15] as well as Aarnborg and Lagergren [4] have concurrently and independently developed $\mathcal{O}(n \log^2 n)$ algorithms for these problems.

In their study of tree decompositions, Robertson and Seymour developed a beautiful structural result which gives a canonical tree decomposition of any graph [31]. Part of this result is a min-max theorem which indicates a relationship between tree-width and separators. For our purposes, a separator in a graph G will be a set X of vertices of G such that no component of $G - X$ contains more than $\frac{2}{3}|V(G - X)|$ vertices. An S-separator for some $S \subseteq V(G)$ will be a set X of vertices of G such that no component of $G - X$ contains more than $\frac{2}{3}|S - X|$ vertices of S. Separators were first studied by Lipton and Tarjan [25] and have been used to develop linear-time algorithms for a number of optimization problems on planar graphs, see [26,27]. The relationship between separators and tree-width is summed up by the following theorem, implicit in [29]:

Theorem 1: *If G has tree width at most k then for any set $S \subseteq V(G)$, G contains an S-separator of order at most k. Conversely, if for every $S \subset V(G)$, G contains an S-separator of order k then G has tree width at most $4k$.*

Using Robertson and Seymour's structural results about tree decompositions, Alon, Seymour, and Thomas [2] managed to extend work of Gilbert, Hutchinson, and Tarjan [19] about separators in graphs of bounded genus to graphs without a order r clique minor for every r. Bienstock, Seymour, and Thomas [9,10,33] use the structural results about tree decompositions to clarify and extend work of Lapaugh, Papadimitriou and others [16,22,23,24,29] on certain graph search problems. Thus, not only do tree decompositions allow us to solve optimization problems quickly in graphs of bounded tree width, they also shed new light on older and well-studied optimization tools and problems.

In this paper, we will discuss, for every fixed k, an algorithm which given a graph G either finds a cutset C of G with $|C| \leq k$ such that no component of $G - C$ has more than $\frac{3}{4}|G - C|$ vertices or determines that G has no $\frac{2}{3}$-separators (the result is presented in this form for simplicity, actually we can do much better). As mentioned above, we can use such an algorithm to determine if a graph on n vertices has tree width at most k and to find a bounded width tree decomposition if one exists. In the next section, we define tree decompositions, sketch Robertson and Seymour's $\mathcal{O}(n^2)$ algorithm for finding a tree decomposition , and indicate how

our approximate separator algorithm can be parachuted into theirs to improve its running time. In Section 3, we give a sketch of our algorithm. In Section 4, we make some concluding remarks.

2. Tree Decompositions and How to Find Them

A tree decomposition $[T, \mathcal{X}]$ for a graph $G = (V, E)$ consists of a tree T and a set $\mathcal{X}$ of subgraphs of G, one for each node of T, such that if we let X_t denote the subgraph of G corresponding to a node t of T then:

(i) $\bigcup_{t \in T} E(X_t) = E(G)$,

(ii) for $s \neq t$, $E(X_s) \cap E(X_t) = \emptyset$, and

(iii) for nodes t_0, t_1, and t_2 of T, if t_2 is on the unique t_0 to t_1 path of T then $V(X_{t_0}) \cap V(X_{t_1}) \subseteq V(X_{t_2})$.

The width of a tree decomposition is $\max_{t \in T}\{|V(X_t)|\}$. The tree width of G is the minimum of the widths of its tree decompositions .

We note that condition (iii) in the above definition ensures that if we root T and for each $t \in T$, let T_t be the subtree of T rooted at t and let $G_t = \bigcup_{s \in T_t} X_s$, then every node of G_t which is incident to some edge of $G - G_t$ is in X_t. This gives some idea of why we can apply dynamic programming to graphs for which we have a tree decomposition of bounded width. We will now describe Robertson and Seymour's algorithm to find such a tree decomposition . Actually, we describe a modification of their algorithm which makes our exposition easier.

To begin, we make the following remarks:

Remark 1: *If G has tree width at most k then $E(G) \leq k|V(G)|$.*

Remark 2: *If G has tree width at most k then for all $S \subseteq V(G)$, G has an S-separator of order k.*

The proofs of these remarks are straightforward and can be found in [32]. Remark 1 implies that we can assume $|E(G)| = \mathcal{O}(|V(G)|)$. Now, given a graph G, our algorithm either finds a set S in G which has no S-separator of order k, or finds a tree decomposition of G of order $4k$. Remark 2 implies that in the first case, G has tree width greater then k. For our purposes, it is convenient to describe the algorithm as solving a slightly more general problem as this allows us to define it recursively. Its specifications follow.

Algorithm k-Tree Finder

Input: *A graph $G = (V, E)$, a subset W of V with $|W| \leq 3k$,*

Output: *Either,*

(i) a tree decomposition $[T, \mathcal{X}]$ of G of width at most $4k$ such that $W \subseteq X_t$ for some $t \in T$, or

(ii) a subset S of G such that G has no S-separator of order k or less.

Description of Algorithm: Attempt to find a W-separator X of order at most k. If no such separator exists, return with output (ii) W. Otherwise, let $U_1,\ldots,U_l$ be the components of $G-X$, let $G_i = X \cup U_i$ and let $W_i = (U_i \cap W) \cup X$. Since X is a W-separator, $|W| \leq 3k$, and $|X| \leq k$, it follows that for each i, $|W_i| \leq 3k$. Thus, we can apply $k-$Tree Finder to (G_i, W_i) for each i. If it turns out that for some i there is an S_i such that G_i has no S_i separator of order at most k then G also has no S_i separator of order at most k so we return (ii) S_i and stop. Otherwise, we find for each i, a tree decomposition $[T_i, \mathcal{X}^i]$ of G_i (we let X_t^i be the element of $\mathcal{X}^i$ corresponding to $t \in T_i$) and a distinguished node t_i of T_i such that $W_i \subseteq X_{t_i}^i$. In this case, we obtain a tree decomposition $[T, \mathcal{X}]$ of G by setting:

(i) $V(T) = \left(\bigcup_{i=1}^{l} V(T_i)\right) \cup \{t\}$,

(ii) $E(T) = \left(\bigcup_{i=1}^{l} E(T_i)\right) \cup \left(\bigcup_{i=1}^{l} \{\overline{tt_i}\}\right)$ and

(iii) $\mathcal{X} = \left(\bigcup_{i=1}^{l} \mathcal{X}^i\right) \cup (X \cup W)$ (that is, $X_t = X \cup W$ and for $s \in T_i$ $X_s = X_s^i$).

It is easy to check that $\mathcal{X}$ is indeed a tree decomposition and that it has width at most $4k$ so we return (i) $(\mathcal{X}, t)$.

□

Now, the key step in this recursive procedure is finding a W-separator of order k. We then separate our problem into l smaller practically disjoint subproblems. It is easy to see that this implies that we will consider at most $\mathcal{O}(n)$ subproblems. In a moment, we will show that we can determine if there is a W-separator of order k in G in linear time. It follows that the total time complexity of our algorithm is $\mathcal{O}(n^2)$. We remark further that if each time we split G into subproblems, the largest has size $(1-\epsilon)n$ for some fixed $\epsilon > 0$ then standard techniques show that the resultant algorithm would actually only take $\mathcal{O}(n \log n)$ time in total. By appending the approximate separator algorithm of the next section to the S-separator algorithm of this section it turns out that we can split our problem into essentially disjoint subproblems all at most three-quarters of the size of the original. If we do this however, we must replace the $3k$ and $4k$ in the description of the algorithm by $4k$ and $5k$ respectively. It remains only to describe how to find an S-separator in linear time, if $|S| \leq 3k$.

First we note that G has the desired small S-separator if and only if there is a partition of G into A,B,X such that $|X| \leq k$,$|A \cap S| \leq \frac{2}{3}|S-X|$, and $|B \cap S| \leq \frac{2}{3}|S-X|$ (Obviously, if such a partition exists then X is an S-separator. Conversely, given an S-separator X we can order the components of $G-X$ as U_1,U_2,...,U_l so that $|U_i \cap S| \geq |U_{i+1} \cap S|$ and then obtain our partition by setting $A = \cup_{i=1}^{j} U_j$ where j is the minimal integer for which $|\cup_{i=1}^{j} S \cap U_j| \geq \frac{2}{3}|S-X|$). So, we will attempt to find such a partition rather than looking for an S-separator directly. We do this by considering all partitions of S into three sets S_A,S_B,S_C with $|S_A| \leq 2|S_B|$, $|S_B| \leq 2|S_C|$ and $S_C \leq k$. We then check, using standard alternating path techniques, wether there are $k+1-|S_C|$ internally vertex disjoint paths from S_A to S_B in $G-S_C$. If there is no such set of paths then we can find a set X of k vertices with $X \cap S = S_C$ such that there are no S_A to S_B paths in $G-X$. It follows that we can find our desired partition of G with S_A in A and S_B in B. Conversely, if the desired partition of G into (A,B,X) exists then upon considering the partition of S with $S_A = A \cap S$, $S_B = S \cap B$, and $S_C = S \cap X$ we will not have the required number of internally vertex disjoint S_A to S_B paths in $G-S_C$ so we will discover some partition of G with the desired properties.

Since S contains at most $3k$ vertices, we consider at most 3^{3k} partitions of S. Since we are looking for at most $k+1$ internally vertex disjoint paths, a standard augmenting path algorithm will either find the paths or the desired cutset for a given partition in $O(k|E(G)|)$ time. It follows that the whole algorithm runs in $O(3^{3k}k|E(G)|)$ time which is linear if we fix k. Furthermore, we remark that we can modify the algorithm so that it gives a split in which each side has at most $(1-r)|S-X|$ vertices for any $r \leq \frac{1}{3}$. We also note that we can also use the algorithm if we have assigned a non-negative integer weight w_v to each vertex v of S and want to find a cutset X such that for each component U of $G-X$, we have $\sum_{v \in S \cap U} w_v \leq \frac{2}{3}\sum_{v \in S-X} w_v$. Finally we remark that this algorithm is linear in $|E(G)|$ and can be applied to any graph not just those of average bounded degree.

3. A Linear Time Algorithm for Approximate Separators

In this section, we discuss how to find our desired approximate separators. We simply want to mimic the algorithm described in the previous section for finding S-separators if S has fixed size. Thus, we will find a representative set $R = \{r_1, \ldots, r_l\}$ such that l is bounded by a function of k (in fact $l \leq 24k$) and such that if we know how a cutset splits the vertices of R then essentially we know how it splits the vertices of G. Given these representative vertices, we can search for an approximate separator for G by considering all possible partitions of R. To be more precise, we find, for some $l \leq 24k$, a partition $\mathcal{T} = \{T_1, \ldots, T_l\}$ of $V(G)$ into l disjoint rooted trees, with roots $\{r_1, \ldots, r_l\}$, such that for any i, every component of $T_i - r_i$ has at most $\frac{n}{24k}$ vertices. This is easy to do via a post-order transversal of a depth-first search tree in which we keep track of the size of the subtrees rooted at each node and repeatedly remove subtrees of size at least $\frac{n}{24k}$ which properly contain no rooted subtrees of this size. Also, for each i, we let $w_i = |V(T_i)|$. We choose the T_i in this fashion because it ensures that the following (easy to prove) lemma is true:

Lemma 2: *For any cutset X of G with $|X| \leq k$ and $X \cap R = \emptyset$, for every component U of $G-X$, $|U - \sum_{r_i \in W} w_i| \leq \frac{n}{24}$.*

Now, using the weighted version of our S-separator algorithm from the last section, we can determine if G has a cutset X such that $|X| \leq k$, $R \cap X = \emptyset$ (we enforce this condition simply by ignoring any partition of R in which $R_C \neq \emptyset$), and for each component U of $G-X$, $\sum_{r_i \in U} w_i \leq \frac{7|G-X|}{24}$.

If we find such a partition then Lemma 2 ensures that no component of $G-X$ has more than $\frac{3n}{4}$ vertices, so X is the required approximate separator. If we do not find such a partition then Lemma 2 ensures that G contains no separator which fails to intersect R. Thus, it remains only to consider the possibility that G contains a separator which intersects R. Since $|R| \leq 24k$ there are a fixed number of possibilities for $R \cap X$ and we can consider these possibilities via a routine but cumbersome induction on k. We shall end this section here and avoiding giving any details of this case as the key ideas in the algorithm are presented in the above discussion.

4.0 Some Remarks

Our algorithm can be modified to obtain a linear time algorithm for the following problem, for each fixed k and r:
Given a graph G, an integer weight w_v for every vertex v of G, and a number $N \geq \frac{2}{3}\sum_{v \in V(G)} w_v$, either find a set X with $|X| \leq k$ such that for every component U of $G - X$ the sum of the weights of the vertices in U is at most $(1+r)N$ or determine that there is no cutset X of G such that for each component U of $G - X$, the sum of the weights of the vertices of U is at most N.
It would be of interest to determine if there is a linear time algorithm which decides if a graph has a separator rather than returning an approximate separator as this algorithm does. Perhaps more challenging is to determine if a graph has tree width at most k(for fixed k).

References

[1] Aho, Hopcroft and Ullman: Design and Analysis of Computer Algorithms, *Addison-Wesley, Reading MA* 1972.

[2] N. Alon, P. Seymour and R. Thomas: A Separator Theorem for Graphs with an Excluded Minor and its Applications, *Proc. 22nd STOC* (1990), 293-299.

[3] S. Arnborg, Jens Lagergren and Detlef Seese: Easy Problems for Tree-Decomposable Graphs, *J. of Algorithms* 12 (1991), 308-340.

[4] S. Arnborg and J. Lagergren: Finding Minimal Mionrs Using A Finite Congruence, *Proc. 18th ICALP(1991)*, 544-555.

[5] S. Arnborg: Efficient algorithms for combinatorial problems on graphs with bounded decomposability - A survey, *BIT 25* (1985), 2-33.

[6] S. Arnborg, D.G. Corneil and A. Proskurowski: Complexity of finding embeddings in a k-tree, *SIAM J. Algebra Discrete Methods* 8 (1987), 277-284.

[7] S. Arnborg and A. Proskurowski: Characterization and recognition of partial 3-trees, *SIAM J. Algebra Discrete Methods* 7 (1986), 305-314.

[8] S. Arnborg and A. Proskurowski: Linear Time algorithms for NP-hard problems on graphs embedded in k-trees, *Discrete Appl. Math.* 23 (1989), 11-24.

[9] D. Bienstock and Paul Seymour: Monotonicity in Graph Searching, *J. of Algorithms* 12 (1991), 239-245.

[10] D. Bienstock, N. Robertson, Paul Seymour and R. Thomas: Quickly excluding a forest, *J. Combin. Theory. Ser. B* 52 (1991), 274-283.

[11] H.L. Bodlaender: Dynamic Programming on Graphs with Bounded Tree-width, MIT/LCS/TR-394, MIT, 1987.

[12] H.L. Bodlaender: NC-Algorithms for Graphs with Bounded Tree-width, RUU-CS-88-4, University

of Utrecht, 1988.

[13] H.L. Bodlaender: Improved Self-reduction Algorithms from Graphs with Bounded Tree-width, RUU-CS-88-29, University of Utrecht, 1988.

[14] H.L. Bodlaender: Polynomial Algorithms for Graph Isomorphism and Chromatic Index on Partial k-trees, *J. of Algorithms* 11 (1990), 631-643.

[15] H. Bodlaender and T. Kloks: Better Algorithms For Path Width And Tree Width, *Proc. 18th ICALP(1991)*, 533-543.

[16] R.L. Breisch: An intuitive approach to speleotopology , *Southwestern Cavers* (published by the Southwestern Region of the National Speleological Society) 6, No.5 (1967), 72-78.

[17] B. Courcelle: Recognizability and second-order definability for sets of finite graphs, preprint, Université de Bordeaux, I-8634, Jan. 1987. See also *Information and Comput.* 85 (1990), 12-75.

[18] B. Courcelle: The Monadic Second Order Logic of Graphs. III. Tree-width, Forbidden Minors, and Complexity Issues, Université de Bordeaux, Report I-8852, 1988.

[19] J.R. Gilbert, J.P. Hutchinson and R.E. Tarjan : A separator theorem for graphs of bounded genus , *J. of Algorithms* 5 (1984), 391-407.

[20] M. Golumbic: Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.

[21] S.T. Hedetniemi: Open problems concerning the theory of algorithms on partial k-trees, working paper, 1987.

[22] L.M. Kirousis and C.H. Papadimitriou: Searching and pebbling , *J. Theoret. Comput. Sci.* 47 (1986), 205-218.

[23] L.M. Kirousis and C.H. Papadimitriou: Interval graphs and searching, *Discrete Math* 55 (1985), 181-184.

[24] A. Lapaugh: Recontamination does not help to search a graph , *Tech. Report* 335, Dept. of Elec. Eng. and Comput. Sci., Princeton University, 1982.

[25] R.J. Lipton and R.E. Tarjan: A separator theorem for planar graphs, *SIAM J. Appl. Math.* 36 (1979), 177-189.

[26] R.J. Lipton and R.E. Tarjan: Applications of a planar separator theorem, *Proc. 18th. FOCS* (1977), 162-170.

[27] R.J. Lipton and R.E. Tarjan: Applications of a planar separator theorem, *SIAM J. Comput.* 9 (1980), 615-627.

[28] J. Matoušek and R. Thomas: On the complexity of finding iso-and other morphisms for partial k-trees, unpublished paper, 1988.

[29] T.D. Parsons: Pursuit-evasion in a graph, *in: Theory and Application of graphs* (Y. Alavi and D.R.

Lick, Eds.), New York/Berlin, pp. 426-441, Springer Verlag, 1976.

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

[31] N. Robertson and P.D. Seymour: Graph minors X. Obstructions to tree decomposition, *J. Combin. Theory Ser. B* 52 (1991), 153-190.

[32] N. Robertson and P.D. Seymour: Graph minors XIII, The disjoint path problem, preprint, Sept. 1986.

[33] P.D. Seymour and R. Thomas: Graph searching, and a minimax theorem for tree-width, submitted (manuscript 1988).

[34] P. Scheffler: Linear-time algorithms for NP-complete problems restricted to partial k-trees, preprint AdW, R-Math-03/87, Karl- Weierstraß Inst. für Math., Berlin, 1987.