NC Algorithms for the Clique Separator Decomposition

Mark B. Novick*

TR 89-974
March 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Research supported by NSF grant CCS-8806979.

NC Algorithms for the Clique Separator Decomposition

Mark B. Novick *

Abstract

We give the first *NC* algorithm for finding a clique separator decomposition of a graph, that is, a series of cliques whose removal disconnects the graph. This algorithm allows one to extend a large body of results which were originally formulated for chordal graphs to other classes of graphs. Our algorithm is a parallel version of Tarjan's sequential algorithm for solving this problem. The decomposition can also be used to find *NC* algorithms for some optimization problems on special families of graphs, assuming these problems can be solved in *NC* for the prime graphs of the decomposition. These optimization problems include: finding a maximum-weight clique, a minimum coloring, a maximum-weight independent set, and a minimum fill-in elimination ordering. We also give the first parallel algorithms for solving these problems by using the clique separator decomposition. Our maximum-weight independent set algorithm applied to chordal graphs yields the most efficient known parallel algorithm for finding a maximum-weight independent set of a chordal graph.

*Department of Computer Science, Cornell University, Ithaca, NY 14853-7501. Supported by NSF grant CCS-8806979

1 Definitions

We let $G = (V, E)$ denote a graph with vertex set V and edge set E. Let n and m denote the number of vertices and edges respectively. In this paper we will assume that G is connected. If X is a subset of V, then $G(X)$ is the subgraph induced by X. When $G(V - X)$ is disconnected we say that X is a *separator* of G. A *clique* of G is a subgraph of G in which every pair of vertices is connected by an edge. We call X a *clique separator* of G if the vertices of X form a clique and X is also a separator of G. Furthermore, X is a *minimal clique separator* of G if it is a clique separator of G and no proper subset of X is a clique separator. A *perfect graph* is one where the maximum clique size equals the chromatic number for every induced subgraph. A *chord* of a cycle is an edge connecting two non-consecutive vertices of the cycle. *Chordal graphs* are graphs where every cycle of length greater than three contains a chord.

2 Introduction

Decompositions are often used in studying graph problems. Many times a graph has a property if and only if the pieces it is decomposed into also have that property. Several graph classes have been characterized by the structure of their clique separators, among them chordal graphs [6,10], path graphs (the intersection graphs of paths in a tree) [15], and Gallai graphs (graphs where every odd cycle of length five or more contains two non-crossing chords) [8]. One of the original motivations for studying the clique separator decomposition is related to the problem of recognizing perfect graphs [9]. If X is a clique separator of G and removing X leaves connected components with vertex sets $V_1, V_2, \ldots, V_k$, then G is perfect if and only if $G(V_1 \cup X), G(V_2 \cup X), \ldots, G(V_k \cup X)$ are all perfect. The clique separator decomposition is a perfection-preserving decomposition.

Decompositions are also useful in designing graph algorithms. The two techniques commonly used with decompositions are divide-and-conquer and dynamic programming. We repeatedly decompose a graph into smaller pieces until we obtain graphs that can no longer be decomposed. These graphs are called *prime graphs* or *atoms*. We solve the problems on these atoms and combine the solutions to find the solutions on larger and larger components until we have the solution for the original graph. Recently, this paradigm has been used to design several efficient parallel and sequential algorithms for graph classes with small separators [1,11].

Suppose X is a clique separator of G. Then there is a vertex partition A, B, X such that no vertex in A is adjacent to one in B. We can then decompose into two components $G' = G(A \cup X)$ and $G'' = G(B \cup X)$, and then recursively perform the decomposition on G' and G'' in turn until only prime graphs result. We can represent the decomposition of G into G' and G'' as a binary tree where the leaves are prime graphs and the internal nodes are clique separators. Following Tarjan's [18] terminology, we call such a tree

a *binary decomposition tree*. Throughout this paper we will assume that each clique separator is a minimal clique separator, because this results in smaller separators and fewer atoms in the decomposition.

In the next section of the paper we show how to efficiently parallelize the sequential algorithms for finding a clique tree decomposition. The fastest published sequential algorithm is due to Tarjan [18], but the algorithm he gave is inherently sequential. This algorithm runs in $O(nm)$ time by first finding a special ordering of the vertices called a *minimal elimination ordering*. We show how a tree representation of the graph can be constructed in parallel, where the edges of the tree correspond to clique separators. Our parallel algorithm runs in $O(\log^2 n)$ time with $O(nm)$ processors assuming we are given a minimal elimination ordering. The processor-time product of our algorithm is nearly optimal. Currently, no NC algorithm is known for finding a minimal elimination ordering, but Klein [14] believes a faster sequential algorithm for finding such an order can be based on his work on parallel chordal graph algorithms. Alternatively, we can parallelize an algorithm that is a variant of one due to Burlet and Fonlupt [3] that runs in $O(n^4)$ time, but their algorithm gives a weaker decomposition. Our NC version uses $O(n^4)$ processors and $O(\log^2 n)$ time with the aid of some chordal graph techniques.

In Section 4 we show how the decomposition can be used to solve several optimization problems for special graph classes. In particular we give NC algorithms for finding a maximum clique, graph coloring, maximum independent set, and a minimum fill-in ordering of a graph, assuming these problems can be solved in NC for the prime graphs in the special graph class. The number of processors used by our parallel algorithms is at most $O(n)$ times the best known sequential running times for these problems. All four of these problems are normally NP-complete, but can be solved in polynomial time for chordal graphs, because the only prime chordal graphs are cliques. Our algorithms first construct a tree representation of the graph and then apply tree processing techniques such as terminal branch removal [16]. We can also find the chromatic polynomial of a graph assuming it is easy to do this on the prime graphs.

3 Parallel Decomposition Algorithms

Whitesides [19] designed the first polynomial time sequential algorithm for finding a clique separator decomposition. Her algorithm ran in $O(n^3m)$ time. Tarjan found a faster algorithm which ran in $O(nm)$ time. Independently of us, Dahlhaus and Karpinski [5,4] have claimed to have an NC algorithm for the problem. Burlet and Fonlupt gave an $O(n^4)$ algorithm which checked whether a graph had a universal clique separator S, one whose removal left each component of the graph with a vertex that was adjacent to all the vertices of S. They used this algorithm to help recognize Meyniel graphs, a class of perfect graphs where every odd cycle contains at least two chords. We will show that their algorithm can be easily modified to find a universal clique separator

decomposition in $O(n^4)$ time. These algorithms are the main sequential algorithms for performing the decomposition. The fastest methods depend on properties of chordal graphs for their efficiency so we will next review the needed results.

First, we define some of the terms used to talk about chordal graphs. An *elimination ordering* π is a numbering of the vertices of G from 1 to n. We define the *fill-in* F_π induced by the ordering π to be the following set of edges:
$F_\pi = \{\{v,w\} | v \neq w, \{v,w\} \notin E$, and there is a path
$v = v_1, v_2, \ldots, v_k = w$ in G such that $\pi(v_i) < min\{\pi(v), \pi(w)\}$
for $i = 2, \ldots, k-1\}$.
A *perfect elimination ordering*, often abbreviated to PEO, is an elimination ordering with no induced fill-in. In a PEO, the higher-numbered neighbors or a vertex form a clique. An ordering π is *minimum* if no other elimination ordering has a smaller cardinality fill-in, and π is *minimal* if there is not an elimination ordering σ such that F_σ is properly included in F_π. The *fill-in graphs* for π is the graph $G_\pi = (V, E \cup F_\pi)$. Tarjan's algorithm depends on the following theorems he proved:

Theorem 1 *Any ordering π is a PEO of G_π.*

Theorem 2 *G has a PEO if and only if G is chordal.*

Theorem 3 *Let π be a minimal ordering. For any decomposition by clique separators, every edge $\{v,w\} \in F_\pi$ is such that a unique atom contains both v and w.*

Our algorithm depends on the following theorem which we prove.

Theorem 4 *Let π be a minimal ordering of G. If C is a clique separator of G_π containing no fill-in edges, then C is also a clique separator of G. Conversely, if C is a clique separator of G, then it is also a clique separator of G_π.*

Proof: The first half of this theorem follows immediately since if C is a clique separator for a graph, then it is a clique separator for any subgraph that contains C and has the same number of vertices as the original graph. The second half follows from the correctness of Tarjan's algorithm. His algorithm checks the cliques induced by a vertex of G_π and its higher-numbered neighbors to determine if they form a clique separator of G. Clearly any clique in G is also a clique of G_π. Any clique separator this algorithm finds in G will also be a clique separator of G_π since π is a PEO of G_π. □

The tree representation we use to parallelize the algorithm is a generalization of the notion of a clique tree of a chordal graph. A *clique tree* of a chordal graph $G = (V, E)$ is a tree T whose nodes are the maximal cliques of G and whose arcs are defined in such a way that the set of maximal cliques containing $v \in V$ form a connected subtree of T. Buneman [2] and Gavril [7] have shown that every chordal graph has a clique tree representation. For an arbitrary graph $G = (V, E)$ we define a *simplicial tree* to be a tree T whose nodes are the atoms of G and whose arcs are defined so that the atoms containing a vertex $v \in V$ form a subtree of T. Efficient parallel algorithms exist for

finding a clique tree in parallel. We will show that the clique separator decomposition of G can be computed in parallel by giving an efficient algorithm for finding a simplicial tree of G. Given the simplicial tree of G, it is easy to find the clique separator decomposition because the intersection of two adjacent atoms in the simplicial tree forms a clique separator of G.

Our algorithm is composed of the following steps:

1. Get a minimal elimination ordering π of G.
2. Form G_π, the fill-in graph induced by π.
3. Form a clique tree T of G_π.
4. Check all pairs of neighboring cliques in T. If the intersection of these two cliques contains an edge in F_π, then the vertices of these two cliques will be in the same atom of G. Merge their vertex sets in this case. On the other hand if the intersection of these two cliques is also a clique in G, then this intersection is a clique separator of G. After all the vertex sets have been merged we are left with a simplicial tree, instead of a clique tree. Note that we are left with $O(n)$ atoms.

Finding a minimal elimination ordering in parallel remains an open problem. The best sequential algorithms for this problem run in time $O(nm)$. Dahlhaus and Karpinski [4] propsed to overcome this difficulty by finding an ordering which preserves the clique separators of the graph, i.e., the original graph and the fill graph induced by the elimination ordering have the same set of clique separators. There are efficient parallel algorithms for the related problem of finding a PEO. Computing the fill-in graph can be done in parallel time $O(\log^2 n)$ time using $O(|E - F_\pi|)$ processors on a CREW-PRAM by using Hafsteinsson's algorithm [9]. Naor et al. [16] gave a $O(n^3)$ processor algorithm for finding a clique tree of a chordal graph, but this can be improved to $O(n + m)$ processors by modifying the elimination tree representation Klein [13] used in his algorithm. Checking if the intersection of two neighboring cliques of G_π contains a fill-in edge takes $O(\log n)$ time on a CREW-PRAM using $O(m - n)$ processors. We find the vertices in the intersection of the two cliques, say there are k of them. If $\binom{k}{2} > m$, then the intersection contains a fill-in edge. Otherwise, check each edge in this clique of size k to determine if it is an edge of G. Performing this computation for all cliques requires $O(nm)$ processors since there are $O(n)$ maximal cliques in G. The merging of cliques can also be done quickly. Assuming we are given a minimal elimination ordering the algorithm takes $O(\log^2 n)$ time on a CREW-PRAM with $O(nm)$ processors.

Burlet and Fonlupt gave a method for finding a universal clique separator of a graph. As stated, their algorithm finds a universal clique separator if one exists, but with slight modifications one can find the decomposition itself in $O(n^4)$ time. The following observation is crucial to this method. Suppose v and w are vertices of G, but

$\{v,w\} \notin E$. Let $U = \{u_1,\ldots,u_j\}$ be the set of vertices adjacent to both v and w. If U induces a clique in G, and removing U from G leaves v and w in different components of the remaining graph, then U is a universal clique separator of G. If G has a universal clique separator, then there is a pair of vertices v and w for which the above method finds then clique separator.

We extend this algorithm to find a simplicial tree of G.

1. Create a graph $H = (V, V \times V)$.

2. For every pair of vertices v,w such that $\{v,w\} \notin E$, compute the set $U = \{u | \{u,v\} \in E, \{u,w\} \in E\}$. Check if U forms a clique in G, whose removal from G leaves v and w in different components, say V_1 and V_2. If such a separating clique exists, remove all edges from H that contain one vertex from V_1 and the other from V_2. Continue doing this for other pairs $v,w \in V$.

3. The graph H that remains will be chordal. Find a clique tree of H. Form a tree where the vertices in each node of the tree are the same as the corresponding nodes of the clique tree of H, and each node also contains the subgraph induced by its contained vertices. This tree will be a simplicial tree of G.

The algorithm requires $O(n^4)$ time because $O(n^2)$ pairs of vertices are examined, and $O(n^2)$ work is done for each vertex pair. Parallelizing the algorithm is easy because what happens for each vertex pair is independent of what happens for the other pairs. Furthermore, finding connected components and clique trees can both be done using only $O(n^2)$ processors in parallel.

Proof: Correctness of the algorithm depends on the following argument. If $A \subset V$ is part of an atom in G, then A induces a clique in H since we cannot separate A with a clique. On the other hand, suppose A is not part of an atom in G. Then A does not induce an atom in H either because any clique that separates A in G also separates it in H. Therefore, the atoms of G have the same vertices as the atoms of H. Furthermore, all the atoms of H are cliques so H is also chordal. The simplicial tree of G corresponds to the clique tree of H because both graphs have the same universal clique separators. □

4 Algorithms for Optimization Problems

Here we show that the clique separator decomposition can be used to find *NC* algorithms for several graph problems. These problems are *NP*-complete for general graphs, but if we can solve these problems efficiently for atoms, then we can solve them quickly for the entire graph. In fact, our first three algorithms only use a linear number of processors.

4.1 Minimum Fill-in Orderings

The problem of determining whether there is an elimination ordering of G that results in k or fewer fill-in edges, is NP-complete if k is a problem parameter [20]. However, if we can find a minimum fill-in ordering for each atom G_i of G, then we can also easily find a minimum fill-in ordering of G. Let F_i be the fill injduced by the ordering on atom G_i. Tarjan proved that $G' = (V, E \cup \bigcup_{i=1}^{k} F_i)$ is chordal. A PEO for G' would also give minimum fill-in for G by yielding minimum fill-in for each G_i. We parallelize the algorithm by computing the PEOs in parallel.

4.2 Maximum Cliques

The maximum-weight clique of G is the maximum of the maximum-weight cliques for each atom of G. If, in parallel, we can find the maximum-weight clique of the atoms of G, then we can also get the largest-weight clique in G.

4.3 Graph Coloring

Suppose there is an efficient parallel algorithm to color the atoms of G. We can use this algorithm to give a parallel algorithm for coloring G. First, find a minimum coloring for each atom of G. Next, form a new graph H with vertex set V. For convenience, we will assume each vertex has a unique number. If there is an atom of G in which vertices v and w receive the same color, and if w is the lowest numbered vertex with this property, then put edge $\{v, w\}$ in H. After H has been formed, find its connected components. The vertices in a component of H will be receive the same color in G. Shrink G by contracting the components of H to single vertices in G. Atoms of G are transformed into cliques of the resulting quotient graph, but both graphs have essentially the same clique separators. Therefore, the quotient graph is chordal since each of its atoms is a clique. In NC, we can optimally color the quotient graph with Klein's coloring algorithm. Finally, color G by assigning each vertex in G the color of the corresponding vertex in the quotient graph. This algorithm can be implemented in NC with the number of processors proportional to the sums of the sizes of the atoms.

4.4 Chromatic Polynomial

For any graph G, the chromatic polynomial $f(G, x)$ is defined to be the number of ways we can color G where we have a choice of x different colors. The chromatic polynomial of K_n is $x(x-1)\ldots(x-n+1)$. Suppose C is a clique whose removal leaves components with vertex sets A and B. Then $f(G, x) = f(G(A \cup C), x) f(G(B \cup C), x)/f(C, x)$. We can color $G(A \cup C)$ and $G(B \cup C)$ independently of each other as long as C receives the

same color in both cases. Given the simplicial tree of G and the chromatic polynomial of each atom, we can compute the chromatic polynomial of G by repeatedly using the above formula at each clique separator of G. Thus the chromatic polynomial of G is the product of the chromatic polynomials of the atoms of G divided by the product of the chromatic polynomials of the simplicial tree clique separators of G. This algorithm parallelizes easily because we can find the clique separators easily in parallel

4.5 Maximum Independent Sets

Given a graph G with integer weights on the vertices, the problem of deciding whether G has an independent set of weight $\geq w$ is NP-complete when w is a problem parameter. Several authors [18,19] have noted that the clique separator decomposition can help solve this problem. Suppose C is a clique of G whose removal leaves connected components induced by vertex sets $A_1, \ldots, A_k$ and B. Denote by $wt(I)$ the total weight of vertex set I, and by $N(v)$ the set of vertices adjacent to v. The following algorithm, a modification of Tarjan's technique, computes the maximum-weight independent set of G.

1. For each $v \in C$ and $j : 1 \leq j \leq k$, find a maximum-weight independent set $I_j(v)$ in $G(A_j - N(v))$. Also find a maximum-weight independent set I'_j in $G(A_j)$.

2. For each vertex $v \in C$, redefine the weight of v to be $wt(v) + \sum_{j:v \in A_j}(wt(I_j(v)) - wt(I'_j))$. Find a maximum-weight independent set I'' in $G(B \cup C)$ with respect to the new weights.

3. Define $I = I'' \cup \bigcup_{1 \leq j \leq k} I'_j$ if $I'' \cap C = \emptyset$. If $v = I'' \cap C$, then define $I = I'' \cup (\bigcup_{j:v \in A_j} I_j(v)) \cup (\bigcup_{j:v \notin A_j} I'_j)$. In either case, $wt(I) = wt(I'') + \sum_j wt(I'_j)$.

We can implement a parallel version of this algorithm by starting out with a simplicial tree representation of G, and arbitrarily choosing one atom to be the root of the tree. Choose the A_j so that each of them is a *terminal branch* of the simplicial tree, that is a path of tree nodes of degree at most two that also includes a leaf node. We can find the maximum-weight independent set of a terminal branch in $O(\log^2 n)$ time with $O(n^3)$ processors by using Hembold and Mayr's parallel algorithm [12] for finding maximum-weight independent sets of interval graphs. Only $O(\log n)$ passes of removing terminal branches is required to process the entire simplicial tree. Also we check only $O(n)$ vertices per clique separator so the whole algorithm requires $O(n^4)$ processors and $O(\log^3 n)$ time, the same as the revised algorithm of Naor et al. [17]. We believe that we can cut down the number of processors to $O(n^3)$ in the case of chordal graphs. In that case each of the atoms of a terminal branch is a clique so we can update the vertex weights with one call maximum independent set per terminal branch, rather than $O(n)$ of them.

5 Conclusion

NC algorithms exist for both performing the clique separator decomposition and applying it to solve hard graph problems. In many ways our results are generalizations of work done on chordal graphs. There has been some debate [13,16] as to what representation of chordal graphs is most appropriate to the design of efficient parallel algorithms. We used both the perfect elimination ordering and clique tree properties of chordal graphs in the design of our algorithms. This suggests that both PEOs and clique trees have their place in the design of parallel algorithms.

References

[1] M. W. Bern, E. L. Lawler, and A. L. Wong. Why certain subgraph computations require only linear time. In *26th FOCS*, pages 117–125, 1985.

[2] Peter Buneman. A characterization of rigid circuit graphs. *Disc. Math.*, 9:205–212, 1974.

[3] M. Burlet and J. Fonlupt. Polynomial algorithm to recognize a Meyniel graph. In C. Berge and V. Chvátal, editors, *Topics on Perfect Graphs*, pages 225–252. North-Holland, 1984. Annals of Discrete Mathematics (21).

[4] Elias Dahlhaus and Marek Karpinski. Efficient parallel algorithm for clique separator decomposition (extended abstract). Technical Report 5831 - CS, Institut Für Informatik Der Universtät Bonn, November 1988.

[5] Elias Dahlhaus and Marek Karpinski. Fast parallel decomposition by clique separators. Technical Report 8525 - CS, Institut Für Informatik Der Universtät Bonn, 1988.

[6] G. A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg*, 25:71–76, 1961.

[7] Fănică Găvril. The intersection graphs of subtrees of trees are exactly the chordal graphs. *J. of Comb. Theory, Series B*, 16:47–56, 1974.

[8] Fănică Găvril. Algorithms on clique separable graphs. *Disc. Math.*, 19:159–165, 1977.

[9] Hjálmtýr Hafsteinsson. *Parallel Sparse Cholesky Factorization.* PhD thesis, Cornell University, 1988.

[10] Andras Hajnal and Janos Surányi. Uber die auflosung von graphen in vollstandige teilgraphen. *Ann. Univ. Sci. Budapest Eotvos, Sect. Math.*, 1:113–121, 1958.

[11] Xin He and Yaacov Yesha. Binary tree algebraic computation and parallel algorithms for simple graphs. *J. of Algorithms*, 9:92–113, 1988.

[12] David Helmbold and Ernst Mayr. Perfect graphs and parallel algorithms. In *1986 International Conference on Parallel Processing*, pages 853–860. IEEE, 1986.

[13] Philip Klein. Efficient parallel algorithms for chordal graphs. In *29th FOCS*, pages 150–161, 1988.

[14] Phillip Klein. personal communication, 1988.

[15] Clyde L. Monma and Victor K. Wei. Intersection graphs of paths in a tree. *J. of Comb. Theory, Series B*, 41:141–181, 1986.

[16] Joesph Naor, Moni Naor, and Alejandro Schäffer. Fast parallel algorithms for chordal graphs. In *19th STOC*, pages 355–364, 1987.

[17] Joesph Naor, Moni Naor, and Alejandro Schäffer. Fast parallel algorithms for chordal graphs. revised and submitted to SIAM J. Computing, 1988.

[18] Robert E. Tarjan. Decomposition by clique separators. *Disc. Math.*, 55:221–232, 1985.

[19] Susan H. Whitesides. A method for solving certain graph recognition and optimization problems with applications to perfect graphs. In C. Berge and V. Chvátal, editors, *Topics on Perfect Graphs*, pages 281–297. North-Holland, 1984. Annals of Discrete Mathematics (21).

[20] Mihalis Yannakakis. Computing the minimum fill-in is *NP*-complete. *SIAM J. Alg. and Disc. Methods*, 2:77–79, 1981.