Parallel Algorithms for the Split Decomposition

Mark B. Novick*

TR 89-1075
December 1989

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

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

Parallel Algorithms for the Split Decomposition

Mark B. Novick *

Abstract

We give a new $O(n^2)$ time algorithm for finding Cunningham's split decomposition of an *arbitrary* undirected graph. We can convert this algorithm to an *NC* algorithm that uses only $O(n^3)$ processors. The related composition operation is a generalization of the modular (also called substitution or X-join) composition. The split decomposition is useful in recognizing special classes of graphs, such as circle graphs, which are the intersection graphs of arcs of a circle, and parity graphs, because these graphs are closed under the inverse composition operation. 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 indecomposable graphs of the decomposition.

A new data structure, which we call a generalized *PQ*-tree, is used to make the algorithm efficient. Generalized *PQ*-trees make it easy to find sets that trivially intersect (one set is contained in the other or they are disjoint) each other. All the calculations on these trees can be done efficiently.

Two other important parts of the algorithm are finding a breadth-first search tree and performing a modular decomposition of a graph. These computations are the bottlenecks to an efficient parallel algorithm since they are the only parts of the algorithm where $\omega(n^2)$ processors are required. However, they can be performed in $O(n^2)$ time sequentially.

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

1 Introduction

Decompositions are often used in studying graph problems. One of the most frequently used ones is the modular decomposition, where we find the modules of a graph, sets of vertices which are connected to the same vertices outside the module. Cunningham [9,8] introduced the split decomposition, which is a symmetric generalization of the modular decomposition. He gave an $O(n^3)$ algorithm for finding the split decomposition of an undirected graph. Ma and Spinrad [12] have found an $O(n^2)$ algorithm. Elsewhere it came up in the study of perfect graphs when Bixby [2] showed that if a graph can be split decomposed into two graphs that are perfect, then the original graph was also perfect. Burlet and Uhry [7] used the split decomposition to recognize parity graphs, and solve problems on them. Bandelt and Mulder [1] showed that a special subclass of the parity graphs, called distance-hereditary graphs, also had a simple characterization in terms of the splits of a graph. Several researchers ([10,6,13]) have used the split decomposition to resolve the long-open problem of how to efficiently recognize circle graphs. These researchers showed that circle graphs are closed under split composition, the indecomposable circle graphs have a unique representation, and these uniquely representable graphs are easily recognized. Bouchet [5,4] has generalized some of these ideas about circle graphs into his theory of isotropic systems.

We give efficient sequential and parallel algorithms for finding the split decomposition of a graph. We generalize some results about the split decomposition of parity graphs (see [7,16]) to graphs in general. Our algorithm first finds the breadth first search (BFS) tree of the graph, and then computes the split decomposition level by level in the BFS tree. To help process each level we modularly decompose the portion of the graph at that level. Extra constraints are present based on how the vertices on neighboring levels are adjacent to each other. We express the possible decompositions and update them by using a new data structure called generalized PQ-trees, hereafter called gPQ-trees.

2 Terminology and Theoretical Background

We say that two sets A and B have a *trivial* intersection if they overlap but neither is contained in the other, and denote this by $A \not\sim B$. We denote A is a subset of B by $A \subseteq B$. If A is a proper subset of B, then we write $A \subset B$.

We use the standard graph theory terminology of Bondy and Murty [3]. Unless stated otherwise G will refer to a connected undirected graph with vertex set $V(G)$ and edge set $E(G)$. A set $U \subseteq V$ defines an *induced subgraph* $G(U)$ whose vertex set is U and contains every edge between these vertices in G. A *clique* or *complete graph* is one where every vertex is mutually adjacent to every other vertex. A *star* is a graph where one vertex, called the *center* is adjacent to the other vertices, called the *spokes*, but no two spokes are adjacent. We let N_x denote the set of vertices which are neighbors of x, namely $\{y \in V : (x,y) \in E\}$. If U is a subset of V, then let $N_x(U)$ denote $\{y \in U : (x,y) \in E\}$. If W is a subset of V, then let $N_W(U)$ denote $\cup_{x \in W} N_x(U)$. The *breadth-first search* (abbreviated BFS) layers of G from a vertex v is a partition of V into sets $L_0, L_1, \ldots, L_p$ so that $x \in L_i$ if the shortest path from v to x has length p. If $1 \leq i < j \leq p$, then let $L_{i:j}$ denote $L_i \cup L_{i+1} \cup \ldots \cup L_j$. Let $P_i^1, P_i^2, \ldots, P_i^q$ be the connected components of the subgraph of G induced by $L_{i:p}$. We will abbreviate P_i^j by P when this is not ambiguous. Let P' denote $P \cap L_{i+1}$, the set of vertices in P in level L_{i+1}.

We repeat the split decomposition terminology defined by Cunningham [8] and introduce some new terms as well. Let G_1, G_2 be graphs having vertex sets $V_1 \cup \{v\}, V_2 \cup \{v\}$ respectively, where $\{V_1, V_2\}$ is a partition of V and $v \notin V$. We define the graph $G = G_1 * G_2$, the *(split) composition* of G_1 and G_2, to have vertex set v and edge set $\{(x,y) : (x,y) \in E(G_1) \cup E(G_2), x \neq v \neq y\} \cup \{(x,y) : (x,v) \in E(G_1) \text{and} (v,y) \in E(G_2)\}$. Note that the composition operation is symmetric.

Split decomposition is the inverse of composition. If $G = G_1 * G_2$ and also $|V_1| \geq 2 \leq |V_2|$, then we say that $\{G_1, G_2\}$ is a *simple decomposition* of G and write $G \rightarrow \{G_1, G_2\}$. We call $\{V_1, V_2\}$ the *split* of G associated with the simple decomposition and v the associated *marker* element. If we remove the restriction $|V_1| \geq 2 \leq |V_2|$, then a *quasi-split* results. The sets V_1 and V_2 are called split sets. A (general) *decomposition* of a graph is defined inductively to be either $\{G\}$ or a set D' of graphs obtained from a decomposition D of G by replacing a member G_1 of D by the members of a simple decomposition of G_1, where the marker of this simple decomposition is not a vertex of any member of D. If D'' is obtained from D by a (nonempty) sequence of operations of the kind described above, then D'' is said to be a *(strict) refinement* of D. If the sequence consists of exactly one operation, the refinement is *simple*.

Two decompositions D, D' of G are *equivalent* if D' can be obtained from D by replacing some of the markers of D by markers of D'. The decomposition D of G is *minimal* with some property P if D has P and no strict refinement of D also has this property. A decomposition is *trivial* if $|D| = 1$. A graph is *prime* if it has no nontrivial decomposition. On the other hand if a graph G has at least four vertices and every partition of $\{V_1, V_2\}$ of V satisfying $|V_1| \geq 2 \leq |V_2|$ is a split, then we say that G is *brittle*.

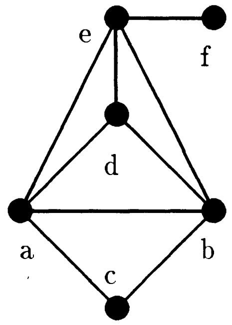
Cunningham's main results are about the uniqueness of decompositions.

Theorem 2.1 *Every connected graph has a unique minimal decomposition (up to uniqueness), each of whose members is prime or brittle.*

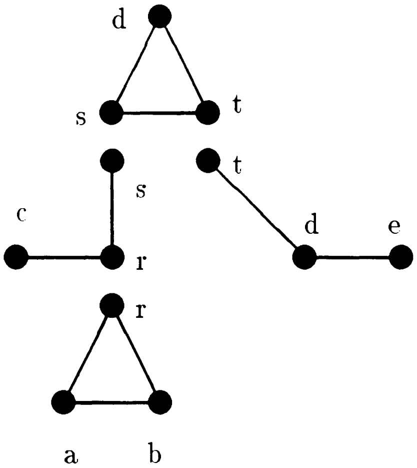
Theorem 2.2 *The only brittle graphs are stars $K_{1,n}$ and complete graphs K_n with at least four vertices.*

This unique minimal decomposition is called the *standard decomposition*.

Cunningham also introduced the notion of a *decomposition tree.* For any decomposition D of a a graph G we define a tree T so that the tree's nodes are the members of D and its arcs are the markers of D. Each marker joins in T the two members of D of which it is a vertex. Our parallel algorithm for the split decomposition produces a decomposition tree of the input graph. See Figure 1 for an example of a graph and its split decomposition tree.



a graph



its decomposition tree

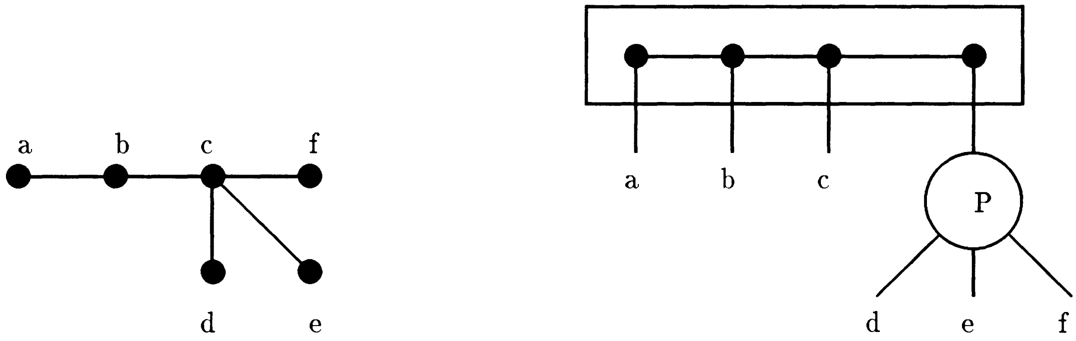
Figure 1: decomposition tree

The *modular decomposition* of a graph G results if we add a new vertex v to G so that v is adjacent to every vertex in G, and then perform a split decomposition of the resulting graph. We root the standard decomposition tree of the resulting graph at the member that contains v. Each vertex of G is a *module.* The *module* induced by a member H of the decomposition tree is the set of vertices of G which are contained in descendants of that member in the decomposition tree. We say that the module is a *series, parallel, or neighborhood module* if H is respectively a clique, a star, or neither. Any vertex outside of a module is either adjacent to all of the vertices in the module or none of them. The *maximal submodules* of H are the modules induced by the children of H in the decomposition tree. We typically write down the modular decomposition tree of a graph by noting which members are series or parallel modules, and in the case of neighborhood modules indicating which maximal submodules contain vertices adjacent to other maximal submodules. See Figure 2 for an example of a graph and its modular decomposition.

gPQ-trees are used to efficiently solve the trivial intersection problem:

- Given a ground set of elements Σ and a collection A of subsets of Σ, which subsets of Σ have a trivial intersection with every set in A?

The leaves of a *gPQ*-tree are the elements of Σ. There are two types of internal nodes



a graph its modular decomposition

Figure 2: modular decomposition

in a *gPQ*-tree, *P*-nodes (denoted by circles) and *Q*-nodes (denoted by rectangles.) Each internal node has at least two children.

Each *gPQ*-tree T represents the subsets of S that have trivial intersections with every element of A. We denote this set of subsets by $M(T)$. The *gPQ*-tree is constructed so that the following subsets of S have trivial intersections with the elements of A:

1. The empty set, S, and all sets which contain only one element of S.
2. The set of leaves which are the descendants of some *Q*-node.
3. For any subset of children of a *P*-node, the set of leaves which are the descendants of this subset of children.

Figure 3 shows *gPQ*-tree reduction in action.

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:

$$\begin{aligned} F_\pi &= \{\{v,w\} | v \neq w, \{v,w\} \notin E, \text{and there is a path} v = v_1, v_2, \ldots, v_k = w \text{in} G \\ &\quad \text{such that} \pi(v_i) < min\{\pi(v), \pi(w)\} \text{for} i = 2, \ldots, k-1\}. \end{aligned}$$

The *fill-in graph* for π is the graph $G_\pi = (V, E \cup F_\pi)$. Note that if $U \subseteq V$ induces a connected component of G, then it also induces a connected component of G_π.

3 Properties of the split decomposition

Consider the BFS layer structure of G rooted at vertex v. Assume for the remainder of he paper that S is a set of vertices in G containing some vertices at distance i from v, but none that are closer. Let $\lambda = L_i \cap S$ in what follows.

Lemma 3.1 *Suppose S is a split set of G. Then for any $x \in V$ at least one of the following must be true: $x \in S$, $N_x(S) = \emptyset$, or $N_x(S) = \lambda$. Conversely, $\{S, V \setminus S\}$ is a split if one of these conditions is met for every vertex v.*

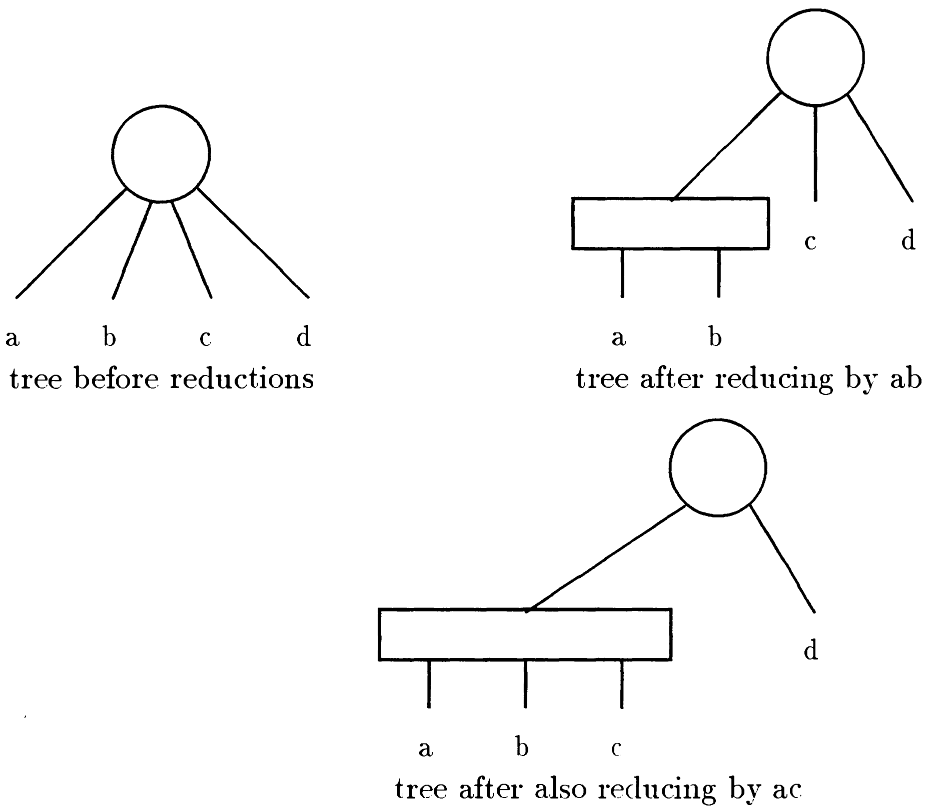


Figure 3: gPQ-tree reduction

Proof: Every vertex in λ is adjacent to a vertex in L_{i-1}. Any two vertices $x, y \in L_{i-1}$ that are adjacent to vertices in S must be adjacent to the same vertices in S, by the definition of a split. However, vertices in L_{i-1} are not adjacent to vertices in L_{i+1}. Therefore, $N_x(S) = N_y(S) = \lambda$. The converse follows trivially from the definition of a split. ∎

In Figure 4 we give the BFS layer structure of a graph, and indicate with a rectangle a possible split based on its BFS tree.

We can examine the situation more carefully by looking at the level of vertex x in this lemma.

Theorem 3.1 *The set S is a split set if the following conditions are met. Conversely, if λ satisfies these four conditions, then there is a split set S satisfying $\lambda = L_i \cap S$.*

1. *If the level of vertex x is different from $i-1$, i, and $i+1$, and $x \notin S$, then $N_x(S) = \emptyset$.*

2. *If $x \in L_{i-1}$, then either $\lambda \cap N_x(L_i) = \emptyset$ or $\lambda \subseteq N_x(L_i)$.*

3. *If $x \in L_i \setminus S$, then either $\lambda \cap N_x(L_i \setminus \{x\}) = \emptyset$ or $\lambda \subseteq N_x(L_i \setminus \{x\})$.*

4. *Suppose $x \in L_{i+1}$ also belongs to P_i^j, a connected component of $L_{i+1} \cup \ldots \cup L_p$. Then one of the following will hold: $\lambda \cap N_x(L_i) = \emptyset$, $\lambda \subseteq N_x(L_i)$, or $N_P(L_i) \subseteq N_P(L_i)$.*

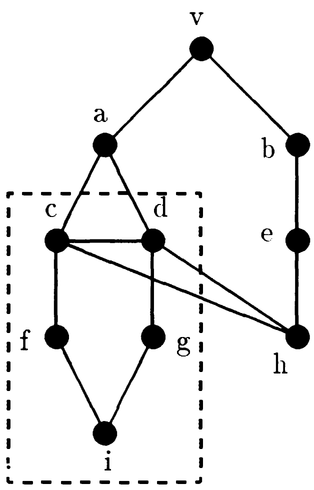


Figure 4: BFS layer structure

Proof: If $x \notin L_{i-1}, L_i, L_{i+1}$ and $x \notin S$, then x has no neighbors in L_i. By Lemma 3.1, this means $N_x(S) = \emptyset$.

We also show that for any $x \notin S$, we have either $\lambda \cap N_x(S) = \emptyset$ or $\lambda \subseteq N_x(S)$. By Lemma 3.1, we know that either $N_x(S)$ is either $\emptyset$ or $L_i \cap S$. If $N_x(S) = \emptyset$, then $(L_i \cap S) \cap N_x(L_i) \subseteq N_x(S) \cap N_x(L_i) = \emptyset$. If $N_x(S) = L_i \cap S$, then $\lambda \subseteq N_x(L_i)$. This handles the case where $x \in L_{i-1}$ or $x \in L_i$.

There cannot exist vertices x and y such that $x \in S \setminus L_i$ and $y \in N_x(L_i) \setminus S$. For if they did exist, then $N_y(S)$ is neither $\emptyset$ nor $L_i \cap S$, contradicting the preceding paragraph. Therefore, if $x \in S \setminus L_i$, then $N_x(L_i) \subseteq L_i \cap S = \lambda$.

If some vertices in P are also in S, then $P \subseteq S$ is required for S to be a split. Taking the union over all $z \in P'$, we see that if some vertex of P is in S, then $N_P(L_i) \subseteq \lambda$. This result finishes off the analysis for the case where $x \in L_{i+1}$.

In the converse we construct a split set S from any λ satisfying the four conditions. In particular we show that for any $x \notin S$ either $N_x(S) = \emptyset$ or $N_x(S) = \lambda$. We construct the set S so that it always contains λ. It also contains every connected component P of $G(L_{i+1:p})$ such that some vertex $w \in P$ is adjacent to some but not all the vertices of λ. Otherwise, S does not contain a vertex in P, and it never contains a vertex in $L_{1:i} \setminus \lambda$.

If $x \notin S$ and $x \notin L_{i-1:i+1}$, then $N_x(L_i) = \emptyset$. Vertices in the first $i-2$ levels are never adjacent to a vertex in S. If a vertex $x \in L_{i+2:p}$ is adjacent to one in S, then $x \in S$, because all of the P_i^k containing it must be in S.

Another possibility is $x \in L_{i-1}$. By condition 2, $\lambda \cap N_x(L_i)$ is equal to either $\emptyset$ or λ. This is equivalent to $N_x(\lambda)$ is either $\emptyset$ or λ, but since x is not adjacent to vertices in $L_{i+1:p}$, we have $N_x(S) = \emptyset$ or λ.

In the third case, $x \in L_i \setminus S$. Once again we have $N_x(\lambda) = \emptyset$ or λ, this time because of condition 3. We show that $N_x(S \setminus L_i) = N_x(S \cap L_{i+1}) = \emptyset$. First of all, x is not adjacent to any vertex in $L_{i+2:p}$. Now suppose $N_x(S \cap L_{i+1}) \neq \emptyset$. Then x is adjacent to some vertex $u \in L_{i+1}$, and all the vertices in u's connected component of $G(L_{i+1:p})$ are also in S. One of these vertices, say w, must be adjacent to some but not all the vertices of λ, else the component P would never have been contained in S. By condition 4, either $N_w(\lambda) = \emptyset$ or λ or $N_P(L_i) \subseteq \lambda$. The latter possibility is ruled out by the fact that $x \in N_P(L_i) \setminus \lambda$, while the former is ruled out by the choice of w. We have a contradiction, so vertex u cannot exist. Therefore, $N_x(S \cap L_{i+1}) = \emptyset$ and $N_x(S) = \emptyset$ or λ.

In the last possibility, $x \in L_{i+1} \setminus S$. Since $x \notin S$, either $N_x(\lambda) = \emptyset$ or λ. Let P be the component of $G(L_{i+1:p})$ to which x belongs. Every vertex in P does not belong to S. Therefore, $N_x(S) = \emptyset$ or λ. ▮

We determine the possible splits of G by answering the question: For which sets $\lambda \in L_i$ is there a quasi-split $\{S, V \setminus S\}$ where $v \in V \setminus S$ and every vertex in S is at a minimum distance of i away from v such that $S \cap L_i = \lambda$? We solve this problem by using gPQ-trees and the modular decomposition to handle the constraints implied by Theorem 3.1.

Corollary 3.1 *If S is a split set, then the following conditions are met. Conversely, if these conditions are met, then there exists a split set S with $\lambda = L_i \cap S$.*

1. *If $x \in L_{i-1}$ or $x \in L_{i+1}$, then $\lambda \not\sim N_x(L_i)$.*
2. *For $x \in L_{i-1}$, $N_x(L_i) \not\subset \lambda$.*
3. *The set λ is a module in $G(L_i)$.*
4. *If $x \in P'$ and $N_x(L_i) \subset \lambda$, then $N_P(L_i) \subseteq \lambda$.*
5. *For every component P in $G(L_{i+1:p})$, $\lambda \not\sim N_P(L_i)$.*

Proof: The first four statements are direct corollaries of Theorem 3.1. The last follows by the following argument. For every $x \in P'$, one of the following is true: $\lambda \cap N_x(L_i) = \emptyset$, $\lambda \subseteq N_x(L_i)$, and $N_P(L_i) \subseteq \lambda$. If the third case never applies, then $N_x(L_i) \cap \lambda$ is $\emptyset$ or λ for every x so that when we take unions, $N_P(L_i) \cap \lambda$ is either $\emptyset$ or λ. Thus $\lambda \not\sim N_P(L_i)$.

We show that if these five conditions hold. then the conditions of Theorem 3.1 also hold, and hence there is a split set set S with $\lambda = S \cap L_i$. Conditions 1 and 2 of the corollary imply condition 2 of the theorem, while the two condition 3's are equivalent. If x is on level $i+1$ and it belongs to component P of $G(L_{i+1:p})$, then by condition 1, one of the following must hold: $\lambda \cap N_x(L_i) = \emptyset$, $\lambda \subseteq N_x(L_i)$, or $N_x(L_i) \subset \lambda$. By condition 4, if $N_x(L_i) \subset \lambda$, then $N_P(L_i) \subseteq \lambda$. Thus condition 2 of Theorem 3.1 is also established, and the argument is complete. ▮

Theorem 3.2 *Suppose P is a connected component of $G(L_{i+1:p})$ which contains at least one vertex in L_{i+1}. Let T be the gPQ-tree representation of L_i after it is reduced by the restrictions implied by Corollary 3.1. Let u be the node of T that satisfies $N_P(L_i) \subseteq leaves(u)$, but no child u' of u satisfies $N_P(L_i) \subseteq leaves(u')$. If every vertex $w \in P'$ satisfies $N_w(L_i) = leaves(u)$, then there is a split set containing $leaves(u)$ but not containing any vertex in $P \cup L_{1:i-1}$. On the other hand, if some vertex in P' is adjacent to some but not all the vertices of $leaves(u)$, then there is no such split set.*

Proof: Suppose the former case applies. Let $\mathcal{Q}$ be the set of connected components of $G(L_{i+1:p})$ such that any connected component $Q \in \mathcal{Q}$ satisfies $\emptyset \subset N_Q(L_i) \subset leaves(u)$. We claim that the set $U = leaves(u) \cup \bigcup_{Q \in \mathcal{Q}} Q$ is a split set. The only vertices in U that are adjacent to vertices outside of U are also in L_i. But any vertex outside of U is either adjacent to every vertex in $U \cap L_i$ or none of them. Hence, U is a split set meeting the desired condition.

On the other hand, suppose there is a vertex $w \in P'$ and $N_w(L_i) \subset leaves(u)$. There is a vertex in L_{i-1} that is adjacent to every vertex of $leaves(u)$. Thus we must include w in any split set which contains $leaves(u)$, but no vertices from the first $i-1$ levels. If we include one vertex from P in the split set, we must include the others as well to preserve the split property. Hence the desired split is impossible. ∎

Corollary 3.2 *A split decomposition tree of the graph G can be found by examining the BFS layering of G. If the components $P_i^1, \dots, P_i^k$ all are spit sets and have the same neighbors in L_i, then the union of any collection of these P_i^j's is also a split set. There is a star member in the decomposition tree. One spoke of this star member corresponds to the root branch of the decomposition tree. Another spoke corresponds to the P_i^j's.*

The gPQ-tree of each of the levels L_i also affects the split decomposition. Each P-node u gives rise to a brittle member of the split decomposition, while each Q-node gives rise to a prime member. In either case this new member contains one more vertex than u has children. One of these vertices is associated with the branch of the graph containing vertices in the first $i-1$ levels. The other vertices are each associated with a child of the gPQ-tree and its descendants.

Two vertices in the same member both not corresponding to the root branch will be adjacent if they correspond to two different children u_1 and u_2 of u, and some vertex in $leaves(u_1)$ is adjacent to some vertex in $leaves(u_2)$. Otherwise, they will not be adjacent. The vertex corresponding to the root branch is always adjacent to the other vertices. If u is a Q-node, then it can give rise to two members in the split decomposition. The first member always occurs and it is a prime member. The second one, if it occurs, is a star whose center is a vertex corresponding to the root branch, and the other vertices correspond to components in $G(L_{i+1:p})$ where every vertex is either adjacent to nothing in L_i or to all the vertices in $leaves(u)$.

If there is a component $P_i^j = P$ which is a split set satisfying $N_P(L_i) = leaves(u)$, then there is also another member in the decomposition tree corresponding to $leaves(u)$. This second member will be a star graph whose center vertex corresponds to $leaves(u)$. One spoke will correspond to the branch containing $L_{1:i} \setminus leaves(u)$. The other branch corresponds to the split sets P satisfying $N_P(L_i) = leaves(u)$.

No further splits of non-brittle members are possible besides the ones handled by the above three cases. The resulting decomposition tree is a (non-strict) refinement of the standard decomposition tree.

These three ways of splitting a graph are illustrated in Figure 5. The decomposition tree of the graph on the left hand side of the figure appears on the right side. The member containing vertices e, f, r appears because the first case applies here. The member containing a, c, s occurs because the second case applies there, and the member containing r, s, t is present because of the third case.

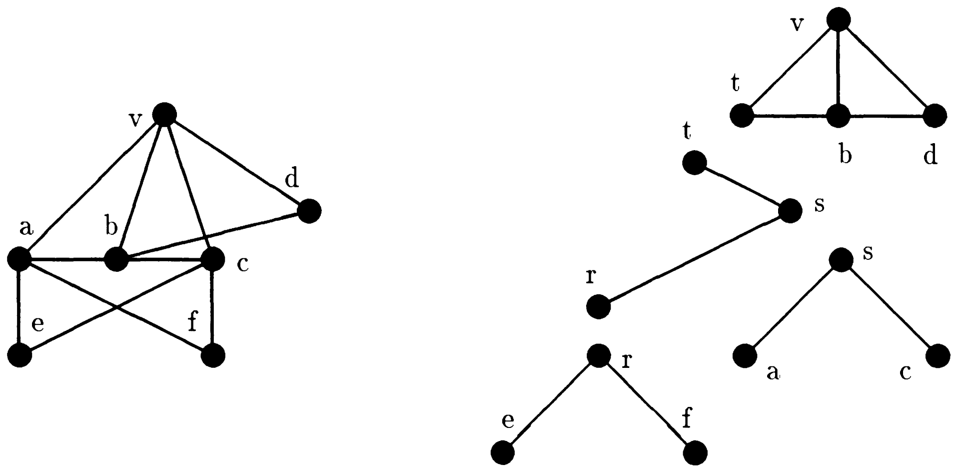


Figure 5: Three ways of forming members of decomposition tree

The first and third cases handle the instances where vertices outside of L_i have the same neighbors in L_i. The second case takes advantage of all the information in the gPQ-tree. P-nodes give rise to brittle members, because the resulting members can be arbitrarily split, whereas the members resulting from Q-nodes cannot be split. No other splits could be possible since we have accounted for all the splits guaranteed by Theorem 3.2.

Given a decomposition tree where every member is either prime or brittle, we can check whether it is a standard decomposition tree, and if not, efficiently convert it to one.

Lemma 3.2 *If T is a decomposition tree where every member is either prime or brittle, then T is a standard decomposition tree if and only if:*

1. *No two members in T sharing a common marker vertex are cliques.*

2. *There does not exist two star members of T sharing a common marker vertex v where one member contains v at the center of the star and the other member contains v at a spoke.*

Proof: If either of these two conditions are violated, we can split compose the two members to form a larger brittle member. In the first case a clique is formed, while in the second a star is born. ▮

Lemma 3.3 *Suppose T is a decomposition tree a graph G, and a and b are two vertices of G in different members of T. Let G_0 be the member containing a, and G_k be the member containing b. The path of members between G_0 and G_k in T will be denoted by $G_1, \ldots, G_{k-1}$. Let $v_i, 1 \leq i \leq k$, be the marker vertices separating G_{i-1} from G_i. Then a and b are adjacent if and only if a is adjacent to v_1, b is adjacent to v_k, and for every i between 1 and k we have v_{i-1} is adjacent to v_i. Furthermore, given T we can determine all the adjacency relations between vertices in G in $O(n^2)$ time sequentially and in NC by using only $O(n^2)$ processors.*

Proof: We can split compose two members sharing a common marker vertex. If we do this until a and b are in the same member, we find that a and b are adjacent only if the above condition is met.

Checking if two vertices are adjacent can also be done efficiently. If two vertices belong to the same member, then answer the adjacency question directly. Otherwise, choose one member of T to be the root of the decomposition tree. Form a tree T' as follows: The nodes of this tree consist of the marker vertices of T. If marker vertex v_1 belongs to member G_0 and its parent G_1, marker vertex, and v_2 in turn belongs to G_2, the parent of G_1, then make v_2 be the parent of v_1 in the tree. Precompute for every pair of marker vertices u_1 and u_2 which vertex is their least common ancestor (abbreviated *lca*) in T, and the marker vertices w_1 and w_2 in the *lca* member such that u_1 and u_2 are descendants of w_1 and w_2 respectively. Construct a forest F of marker vertices where two vertices are adjacent if they were adjacent in T' and they are adjacent in the common member which contains them.

The analysis now boils down to two cases.

1. The member containing a is a descendant of the one containing b.

   Let u be the marker vertex which belongs to the same member as a, and also belongs to the parent of this member. Let w be the marker vertex which is an ancestor of u in T' and also belongs to the member containing b. Then a and b are adjacent if and only if a is adjacent to u, b is adjacent to w, and u and w belong to the same connected component of F.

2. The member containing a is neither an ancestor nor a descendant of the one containing b.

   Let u_1 be the marker vertex which belongs to the same member as a, and also belongs to the parent of this member. Let w_1 be the marker vertex which is an ancestor of u_1 in T' and also belongs to the lca member of a and b. Let u_2 and w_2 denote the corresponding vertices for b. Then a and b are adjacent if and only if a is adjacent to u_1, u_1 and w_1 belong to the same component of F, w_1 is adjacent to w_2, w_2 and u_2 belong to the same component of F, and u_2 is adjacent to b.

If we split compose the members between the one containing a and the one containing b, then we see that they are adjacent only if the above conditions are fulfilled. Every adjacency calculation can be done in constant time, and there are only $O(n^2)$ values to be precomputed. Therefore, all the adjacency checks can be performed efficiently as claimed. ▮

In Figure 1, vertices a and d are adjacent, while a and e are not. This is what we would expect from Lemma 3.3. There are edges ar, rs, st, and td in the decomposition tree of Figure 1, so a and d are adjacent. There is not edge te there. Hence a and e are not adjacent.

Theorem 3.3 *A decomposition tree containing only prime and brittle members can be turned into a standard decomposition tree in $O(n^2)$ time sequentially or in NC by using only $O(n^2)$ processors.*

Proof: Split compose brittle members whose composition is also brittle. Determine the adjacency relations in the members of the new decomposition by applying Lemma 3.3. ▮

4 Finding the split decomposition

The algorithm to find the split decomposition of a graph is based on Corollary 3.2, which relates split sets and a gPQ-tree of vertices in L_i. Find the maximal subsets λ of L_i that satisfy Corollary 3.1. Introduce a marker vertex for each of these λ's. Also introduce marker vertices corresponding to each split which corresponds to a node of the gPQ-tree. (In these splits S contains λ while $V \setminus S$ contains v and $L_i \setminus \lambda$.) The vertices in component $P = P_i^j$ appear in a member of the split decomposition which contains the marker vertex corresponding to the lca in the gPQ-tree of the vertices in $N_P(L_i)$.

1. Choose a vertex v and partition the vertices of G into levels $N_0, \ldots, N_p$ according to the distance of the given vertex to vertex v.

2. For each i find sets $\{P_i^j\}$.

3. Find the modular decomposition of the graphs induced by each L_i. Construct a gPQ-tree corresponding to these modular decompositions.

4. For every vertex $x \in L_{i-1}$ or L_{i+1}, reduce the i-th tree by $N_x(L_i)$.

5. For every connected component P of the subgraph induced by $L_{i+1:p}$, reduce the i-th tree by $N_P(L_i)$.

6. Further reduce the i-th gPQ-tree by enforcing the following conditions:

   (a) For every vertex $x \in L_{i-1}$, $N_x(L_i) \not\subset L_i \cap S$.

   (b) For every vertex $x \in L_{i+1}$ but x is in connected component P of the subgraph induced by $L_{i+1:p}$, then it is not the case that $N_x(L_i) \subset L_i \cap S \subset N_P(L_i)$.

7. Find the permissible splits by looking at the gPQ-tree.

8. Construct a decomposition tree corresponding to these splits. Turn this tree into the standard decomposition tree.

The implementation of the above steps is as follows:

Step 1. BFS can be done sequentially in $O(m+n)$ time. In parallel it can be done in $O(\log^2 n)$ time using only $M(n)$ processors.

Step 2. Order the vertices so that those at the highest levels come first. Use this order as an elimination order, and form the resulting filled graph $G' = (V, E')$. There is a sequential algorithm [17] to do this which runs in $O(|E'|)$ time . Hjafsteinsson [11] has given an NC algorithm for this problem which uses $O(|E'|)$ processors and $O(\log^2 n)$ time. Each P_i^j forms a connected component of $G'(L_{i+1:p})$. In fact, P' induces a connected component of $G'(L_{i+1})$. If u and w are two vertices in L_{i+1}, then we can convert any path between them in $G(L_{i+1:p})$ into a path between them in $G'(L_{i+1})$.

Step 3. Muller and Spinrad's algorithm for modular decomposition runs in $O(n^2)$ time. Novick [14] has given a parallel version which uses $O(n^3)$ processors and $O(\log n)$ time. To turn the modular decomposition into a gPQ-tree, just replace each series and parallel module with a P-node, and each neighborhood module with a Q-node. This operation can be carried out easily, both sequentially and in parallel.

Steps 4 through 6. We use Novick's [15] gPQ-tree routines here. We are reducing the tree corresponding to the i-th level with $O(n)$ sets. Therefore, the sequential algorithm does $O(n|L_i|)$ work for this level. Summing up over each level, we see that the algorithm runs in $O(n^2)$ time. Similarly, by using $O(n^2)$ processors in parallel in $O(\log^2 n)$ time we can perform the reductions.

Steps 7 and 8. By using Corollary 3.2, we find the permissible splits at each layer of the BFS tree. We can find pairs of vertices and/or components with the same neighbor sets by considering the list of neighbors as a bit string, and then sorting the bit strings.

Since we sort $O(n)$ bit strings of length $O(n)$, the sorting process can be done by using $O(n^2)$ processors in $O(\log n)$ time. In a sequential version, bucket sort can be used, and an $O(n^2)$ time algorithm results for these steps.

The process of finding the splits can most easily understood by looking at an example, like the one in Figure 6. Find the P_i^j's in the graph.

$$\begin{array}{ll} P'^1_1 & \{c,d,e\} \\ P'^2_1 & \{f,g\} \\ P^1_2 & \{h\} \\ P^2_2 & \{i\} \\ P^3_2 & \{j,k,l\} \end{array}$$

The vertices c, d form a module in L_2. Before the gPQ-tree is refined in step 5, the gPQ-tree of P'^1_1 contains a two internal nodes, one of which is the parent of the other. The latter node has children c and d. After refinement in Step 5, the resulting tree contains a Q-node, say u, with children c,d,and e. Components P^1_2 and P^2_2 both are adjacent to all the vertices in $leaves(u)$. By case 1 of Corollary 3.2 the decomposition tree contains the member with vertices r, h, i. By case 3 the tree also includes the star member with vertices r, s, t. The resulting decomposition tree is also shown in Figure 6. If we were to carry out the algorithm at the first level also, then the member containing a, b, f, g, t, v would also be split.

Theorem 4.1 *There exists an $O(n^2)$ time sequential algorithm for finding the split decomposition of an undirected graph. This algorithm can be parallelized to run in $O(\log^2 n)$ time on a CRCW PRAM using $O(n^3)$ processors.*

References

[1] H. J. Bandelt and H. M. Mulder. Distance-hereditary graphs. *J. of Combin. Theory, Series B*, 41:182–208, 1986.

[2] R. E. Bixby. A composition for perfect graphs. In C. Berge and V. Chvátal, editors, *Topics on Perfect Graphs*, pages 221–224. North-Holland, 1984. Annals of Discrete Mathematics (21).

[3] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North-Holland, New York, 1976.

[4] A. Bouchet. Digraph decompositions and Eulerian systems. *SIAM J. Alg. and Disc. Methods*, 8:323–337, 1987.

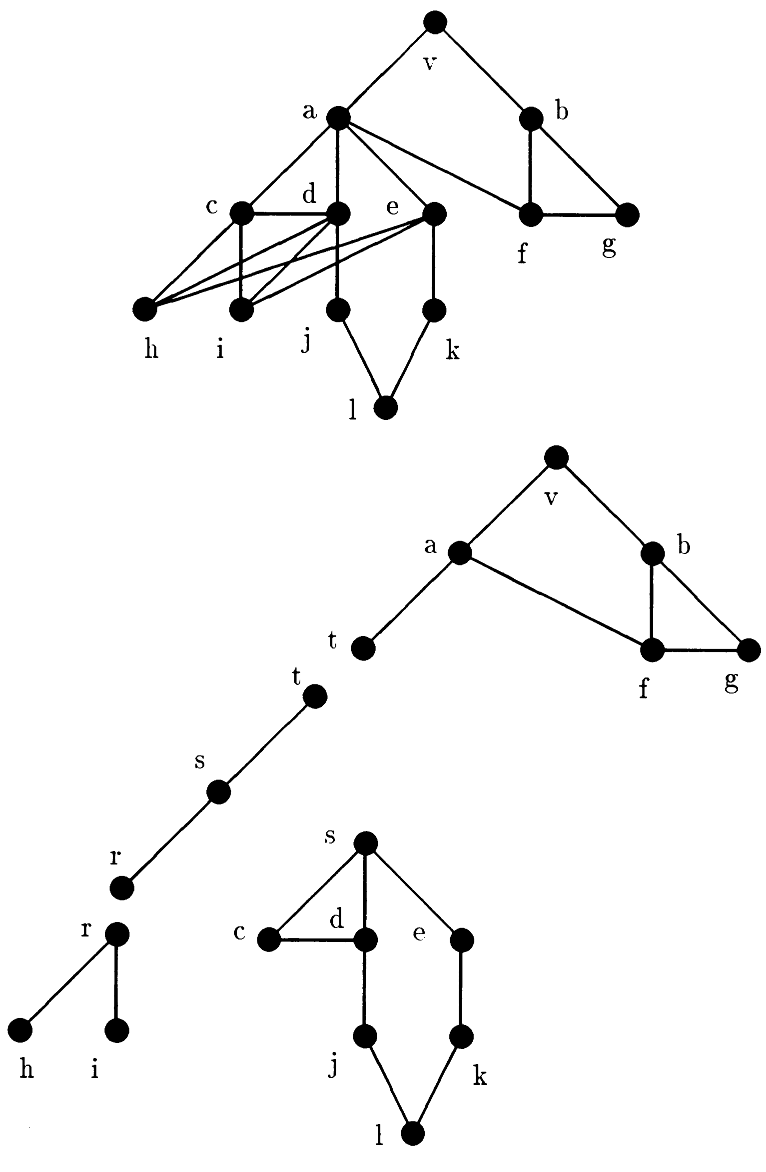


Figure 6: algorithm example

[5] A. Bouchet. Isotropic systems. *European J. Combin.*, 8:231–244, 1987.

[6] A. Bouchet. Reducing prime graphs and recognizing circle graphs. *Combinatorica*, 7:243–254, 1987.

[7] M. Burlet and J.-P. Uhry. Parity graphs. In C. Berge and V. Chvátal, editors, *Topics on Perfect Graphs*, pages 253–277. North-Holland, 1984. Annals of Discrete Mathematics (21).

[8] W. H. Cunningham. Decomposition of directed graphs. *SIAM J. Alg. and Disc. Methods*, 3:214–228, 1982.

[9] W. H. Cunningham and J. Edmonds. A combinatorial decomposition theory. *Canad. J. Math.*, 32:734–765, 1980.

[10] C. P. Gabor, W.-L. Hsu, and K. J. Supowit. Recognizing circle graphs in polynomial time. *J. of Assoc. Comput. Mach.*, 36:435–474, 1989.

[11] H. Hafsteinsson. *Parallel Sparse Cholesky Factorization*. PhD thesis, Cornell University, 1988.

[12] T.-H. Ma and J. Spinrad. Split decomposition of undirected graphs. In *1st Symposium on Discrete Algorithms*, 1990.

[13] W. Naji. Graphes des cordes. caractérisation et reconnaissance. *Disc. Math.*, 54:329–337, 1985.

[14] M. B. Novick. Fast parallel algorithms for the modular decomposition. Technical Report TR89-1016, Cornell University, 1989.

[15] M. B. Novick. Generalized *pq*-trees, 1990. forthcoming tech report.

[16] T. Przytycka and D. Corneil. Parallel algorithms for parity graphs. manuscript, 1989.

[17] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.