Discrete Mathematics and Theoretical Computer Science **5**, 2002, 55–70

Graph Decompositions and Factorizing Permutations

Christian Capelle, Michel Habib and Fabien de Montgolfier

LIRMM – UMR 5506 Université Montpellier II et CNRS
161, rue Ada – 34 392 Montpellier Cedex 05 - France
email: {capelle,habib,montgolfier}@lirmm.fr

received Jul 16, 1998, revised Aug 22, 2000, accepted Sep 22, 2001.

A factorizing permutation of a given graph is simply a permutation of its vertices of which all decomposition sets are factors [3]. Such a concept appears to play a central role in recent papers dealing with graph decomposition. It is applied here for modular decomposition of directed graphs, and we propose a simple linear algorithm that computes the whole decomposition tree when a factorizing permutation is provided.

The approach of using parenthesized factors of a list was first used by Hopcroft and Tarjan for triconnected components search [14]. Our algorithm can be seen as a common generalization of Hsu and Ma [16, 15] for modular decomposition of chordal graphs and Habib, Huchard and Spinrad [10] for inheritance graphs decomposition [1]. It also suggests many new decomposition algorithms for various notions of graph decompositions.

Keywords: Graph algorithms, graph decompositions, modular decomposition.

1 General decomposition framework

Many optimization methods for graphs begin with some decomposition techniques, using the classic divide and conquer paradigm. We restrict our study to decompositions that lead to a decomposition tree of the set of vertices. For a given graph $G = (X, E)$, such a decomposition tree T_G is defined as follows: the vertices of G are in one-to-one correspondence with the leaves of T_G, and a node of the tree correspond to a subset of X: the leaves that descend from it. Therefore a node of T_G may be identified with the subset of vertices it induces. The root is then the vertex set X itself. If a node N of T_G has children $N_1, \ldots, N_k$, this means that the induced subgraph G_N admits the decomposition $G_{N_1}, \ldots, G_{N_k}$. Let us call the nodes of a decomposition tree (and the sets of vertices they induce) ***decomposition sets***.

The existence of such decomposition trees is the consequence of uniqueness decomposition theorems (see Cunningham and Edmonds [5] for a general theory on these combinatorial decompositions) and in the following we will assume that the decomposition trees we construct are uniquely defined up to isomorphism.

This paper deals with the notion of a ***factorizing permutation*** σ of the vertex set X, and it will be convenient to consider σ as a word $\sigma(1)\sigma(2)\ldots\sigma(n)$.

Definition 1 *A permutation σ of the vertex set X is called a factorizing permutation for a given decomposition if every decomposition set is a factor of σ (i.e. its vertices appear consecutively in σ).*

Such a permutation always exists as soon as the decomposition tree is provided, since it can be obtained by a simple preorder traversal of the tree. In this paper we consider the inverse problem.

Central Problem:

Data: a graph $G = (X, E)$ and a factorizing permutation σ.
Find: the decomposition tree (or the decomposition sets).

For a leading example of such a graph decomposition, one can take the ***modular decomposition***, also called ***substitution decomposition***, of graphs (an overview of this theory and its applications can be found in [21]). We present the basic concepts of this decomposition in the following section. In this article we deal with the general case of decomposition of directed graphs. Other examples are modules or blocks for inheritance acyclic directed graphs leading also to decomposition trees [17, 10, 2, 1].

Clearly it could be hard to find a factorizing permutation, but in some cases a factorizing permutation is given for free. As for example as noticed by Hsu and Ma [16, 15] in the case of chordal graphs, the cardinality lexicographic breadth first search of the graph yields a factorizing permutation for modular decomposition. Similarly, as noticed by Ducournau and Habib [8], any depth-first greedy linear extension of an inheritance graph yields a factorizing permutation for the module decomposition of inheritance graphs.

Moreover Habib and Paul [12] have proposed a very simple $O(n+m)$ algorithm to compute a factorizing permutation of a cograph. This work has been generalized to modular decomposition [13] with an $O(n + m\log n)$ algorithm for undirected graphs. These algorithms are easy to understand. As in the previous examples, they show the usefulness of dividing the calculation of the decomposition tree in two steps : calculation of the factorizing permutation, and calculation of the tree from this permutation.

Our algorithm deals with the modular decomposition of graphs (directed or not), but it can easily be adapted for the other decompositions mentioned above for inheritance graphs. It runs in $O(n+m)$, and therefore is a common generalization of Hsu and Ma [16] and Habib *et al.* [10] which were relatively sophisticated *ad hoc* algorithms. Here we propose a simpler algorithm.

As a consequence, such an algorithm and the notion of factorizing permutation introduce some new perspectives for graph decomposition algorithms. Indeed, the decomposition process can be divided into two steps. First, find a way to produce a factorizing permutation, and then use the general algorithm defined here to compute the decomposition tree. Therefore one can focus on the search of such factorizing permutations.

Our central problem is connected to the search of the tree structure involved in factorizing permutation, and therefore a problem of interest by its own. One of the main results of this work is to propose a linear algorithm that applies both for directed or undirected graphs. Furthermore, other graph decompositions (for example those associated with edge-partitioning) can also be considered using this approach [2].

Another example of this paradigm could be found in Hopcroft and Tarjan's pioneering work. Namely their algorithm [14] for triconnected graph recognition uses two depth-first searches and collect triconnected components by "factors".

This paper is organized as follows. The next section presents main results about Modular Decomposition. In Section 3 we introduce a new object, the fracture tree, and present its properties. In Section 4 we propose a simple algorithm leading from the fracture tree to the modular decomposition tree, and its linearity is proven in Section 5. In Section 6 (Conclusion) several directions for further research and a conjecture are presented.

2 Modular decomposition

The modular decomposition (also called substitution decomposition) is very important since its study plays a central role in the area of partial orders, comparability graphs, and transitive orientations [9]. Here we focus on decomposition sets that are called strong modules, which are defined as follows:

Definition 2 *Two set E and F* **overlap** *if $E \cap F \neq \emptyset$, $E \setminus F \neq \emptyset$, and $F \setminus E \neq \emptyset$. Let $G = (X, E)$ be a graph with n vertices and m arcs. The outer neighborhood of v is written $\Gamma^+(v)$ and its inner neighborhood is $\Gamma^-(v)$. A subset M of X is a* **module** *if $\forall v \in X \setminus M$, M does not overlap neither $\Gamma^+(v)$ nor $\Gamma^-(v)$.*

Every set with 1 or n vertices is a ***trivial*** module. A graph with no non-trivial module is ***prime***. A ***strong module*** is a module that does not overlap any other module. They are the decomposition sets of the modular decomposition. In the following a ***factorizing permutation*** is considered with respect to the strong modules. Notice that a given graph may have many factorizing permutations, up to $n!$ for prime graphs.

The ***Modular decomposition tree*** of a graph G is the transitive reduction of the inclusion order of the strong modules of G.

Each node N ($N \subset X$ is the strong module represented by this node) of the decomposition tree is labeled by the quotient graph of G_N according to its children. Such a quotient graph is called the ***representative graph*** of the node N.

Four mutually exclusive cases can be distinguished :

- A node is labeled ***series*** if its representative graph is a clique.
- A node is labeled ***parallel*** if its representative graph is a stable set.
- A node is labeled ***order*** if its representative graph is a total ordering.
- If no one of the previous cases apply, a node N is labelled ***prime***. The representative graph of N is a prime graph.

Let us recall the well-known decomposition theorem (for example see [21])

Theorem 1 (Modular Decomposition) *Let $G = (X, E)$ be a digraph such that $|X| \geqslant 2$. One of the four following claims is true:*

1. *G is not connected and it can be decomposed according to its connected components. The corresponding node in T_G will be a parallel node.*
2. *$\overline{G}$ is not connected and G can be decomposed according to the connected components of $\overline{G}$. The corresponding node in T_G will be a series node.*

3. *G can be decomposed in strong modules such that the quotient graph is a non-trivial total ordering. The corresponding node in T_G will be an order node.*
4. *Else, the maximal non-trivial strong modules of G do not overlap. The quotient graph is prime and the corresponding node in T_G will be a prime node.*

Even though a given graph could have exponentially many modules, it has $O(n)$ strong modules. Therefore an algorithm has to deal only with the good ones, the strong modules.

Let us denote by $LCA(x,y)$ the Least Common Ancestor of the vertices x and y in the modular decomposition tree. The following property trivially holds:

Property 1 *$LCA(x,y)$ is a series node $\Rightarrow (x,y) \in E$ and $(y,x) \in E$*
$LCA(x,y)$ is an order node $\Rightarrow$ either $(x,y) \in E$ or $(y,x) \in E$
$LCA(x,y)$ is a parallel node $\Rightarrow (x,y) \notin E$ and $(y,x) \notin E$

Although linear decomposition algorithms are now available [4, 20, 7], they remain rather hard to implement, and it is still worthwhile to search for simplifications. The result presented here can be understood as a step forward in this direction. Moreover, existing algorithms deal only with undirected graphs, while ours deals with both directed and undirected graphs. The next section presents an interesting decomposition tool; namely the fracture tree.

3 The Fracture Tree

3.1 Notations and definitions

Notations In the following, $G = (X,E)$ is a graph (directed or not) with n vertices and m arcs, and $\sigma(G)$ (or σ if unambiguous) a factorizing permutation of G. The successor of x in σ is x' and its predecessor is $'x$. If $\sigma^{-1}(x) < \sigma^{-1}(y)$ then x is to the left of y. If G is undirected, it is treated as a directed symmetric graph, so the same notations are used for both cases.

Definition 3 *Given a set $S \subset X$ of vertices, a vertex x is a* **cutter** *of S if S is not a module of the induced graph $G(S \cup \{x\})$. This means that $x \notin S$ and that at least two vertices of S have different adjacency relations with x. The set of cutters of S is denoted by $Cut(S)$.*

Of course M is a module if and only if $Cut(M) = \emptyset$. Furthermore we have

Property 2 *let M be a module and S a subset of M, then $Cut(S) \subset M$.*

Since σ is supposed to be a factorizing permutation, the strong modules are factors of it, so the cutters "gather together" in the vicinity of any factor S, in a manner shown below. In order to reach linear time for our algorithm, the only subsets we deal with are the ***pairs***, made up with two *consecutive* vertices of σ. Fortunately there is no need of computing the set of all cutters of a given pair: it is enough to consider the furthest ones.

Definition 4 *Let $P = \{x,x'\}$ be a pair of vertices. The* **first cutter** *of P, denoted by $fc(P)$, is the leftmost vertex of $Cut(P)$ (if any) appearing in the factor $\sigma(1)\ldots x$. If such a vertex exists, the* **left fracture** *of P, denoted by $Lf(P)$, is the factor $fc(P)\ldots x$. The* **last cutter** *of P, $lc(P)$, is the rightmost vertex of $Cut(P)$ appearing in $x'\ldots\sigma(n)$ and the* **right fracture***, $Rf(P)$, is the factor $x'\ldots lc(P)$.*

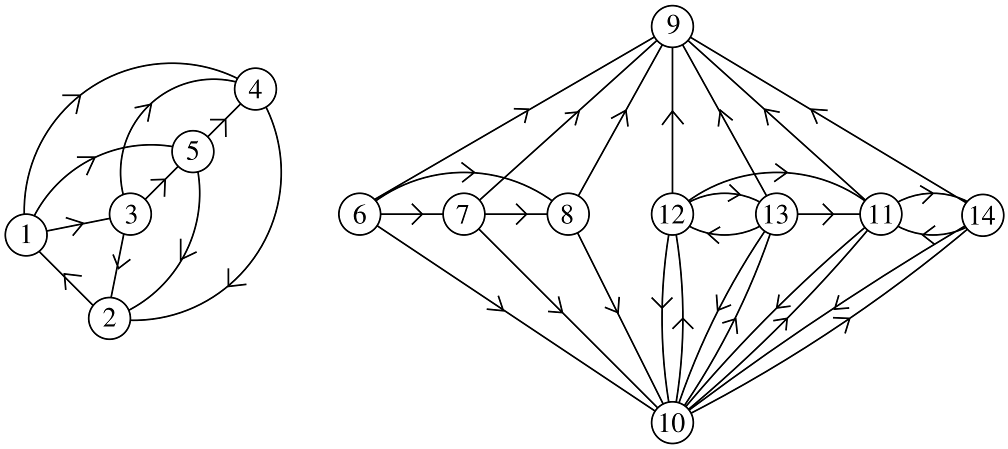


Fig. 1: A directed graph

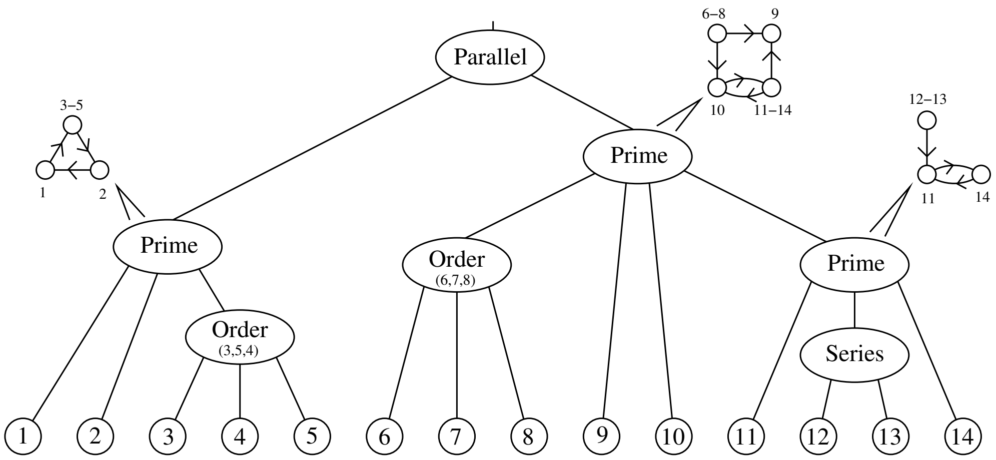


Fig. 2: Its modular decomposition tree

3.2 Parenthesizing

Since a fracture is a factor of σ, it can be enclosed into parentheses. The ***parenthesized factorizing permutation*** $\hat{\sigma}$ is the word where such parentheses have been inserted: an opening parenthesis to the left of $fc(\{x,x'\})$ and a closing one to the right of x mark the left fracture, and likewise for the right fracture. If many opening and closing parentheses have to be written between two consecutive vertices, all the closing ones must be inserted before the first opening one. Finally the permutation itself is enclosed with two extra parentheses.

$$\hat{\sigma} = (\overset{5}{(}\overset{2}{(}1\overset{1}{(}2\overset{2}{)}\overset{2}{(}3\overset{3}{(}4\,5\overset{3}{)}\overset{2}{)}\overset{1}{)}\overset{5}{)}\overset{5}{(}\overset{10}{(}6\,7\,8\overset{8}{(}9\overset{9}{(}10\overset{10}{)}\overset{5}{)}\overset{10}{(}\overset{13}{(}11\overset{11}{(}12\,13\overset{13}{)}\overset{10}{)}14\overset{11}{)}\overset{9}{)}\overset{8}{)})$$

Fig. 3: Parenthesized factorizing permutation of the graph Figure 1, with $\sigma = 1, \ldots, 14$. The exponent written upon a parenthesis just denotes the first vertex of the pair for which this parenthesis marks the fracture.

We claim that $\hat{\sigma}$ is a well-formed word of parentheses, also called a ***Dyck word***, because an equal number of opening and closing parentheses are inserted, and furthermore the opening parenthesis of a fracture appears always before its closing one.

Fractures have the following property:

Property 3 *A module and a fracture can not overlap.*

Proof: Let M be a strong module, F a fracture (left or right) and $P = \{x,x'\}$ the pair of vertices that create F. By Property 2, if $P \subset M$ then $Cut(P) \subset M$, so any fracture of P is a factor of M. If $x \in M$ and $x' \notin M$, then the right fracture of P does not intersect M, and its left one has the same right boundary as M and thus can not overlap it. If $x \notin M$ and $x' \in M$, the case is symmetric. At least if neither x nor x' belongs to M, since M is a module either all or none of its vertices cuts P, and since M is a factor of σ it can not overlap a fracture of P. □

Lemma 1 *Let M be a strong module of G and $\hat{\sigma}$ a parenthesized factorizing permutation of G. $\hat{\sigma}[M]$ is a Dyck subword of $\hat{\sigma}$.*

Proof: Let us *color in red* the fracture parentheses of the inner pairs of M. Property 2 claims those fractures to be factors of M. Furthermore the right fracture of the pair formed with the first vertex of M and its predecessor, and the left fracture of the pair formed with the last vertex of M and its successor, are also colored in red only if those fractures are factors of M. All the remaining parentheses are left in black. The corresponding fractures have at least one vertex out of M, and Property 3 ensures that they do not overlap M, so the black parentheses are out of M. We can swap red and black parentheses on the boundaries of M so that red parentheses are a factor of $\hat{\sigma}$. It made up with pairs of fracture parentheses, so it is a Dyck word. □

3.3 The fracture tree

Given a word of parentheses, a natural idea is to construct the associated tree, using a simple pushdown automaton. Let us call ***fracture tree*** $FT(\sigma)$ the tree of the parenthesized factorizing permutation $\hat{\sigma}$. Its leaves are the vertices of G, which are the letters of σ enclosed in the parenthesizing.

The two extra parentheses correspond to the root of the decomposition tree, and the relationship between $\hat{\sigma}$ and the fracture tree is then straightforward.

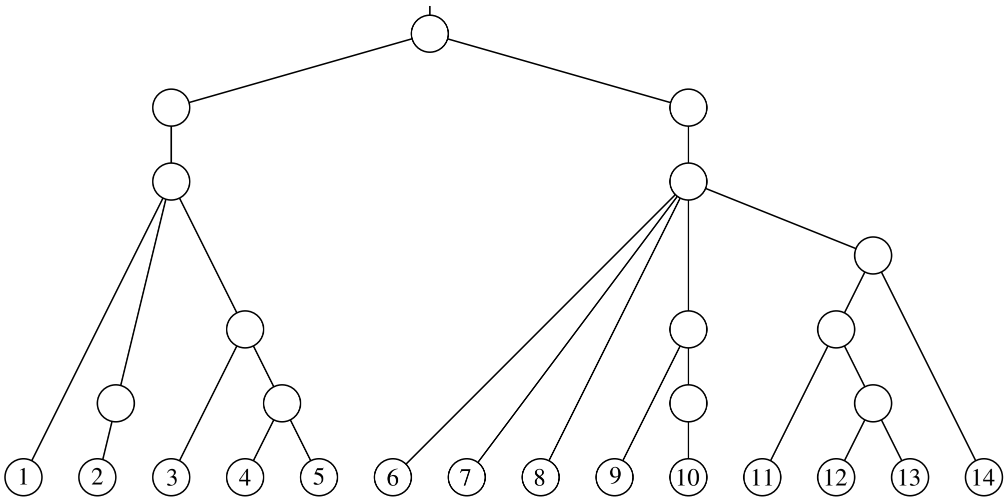


Fig. 4: Fracture tree of our example

The tree can be build from left to right using the following automaton that scans the Dyck word from left to right, start from a special vertex $CurrentNode = Root$ and applying the following rules when a symbol (,) or a letter x is read :

(: add a new vertex child N to the current node, $CurrentNode = N$;

) : backtrack to the father of $CurrentNode$, $CurrentNode = Father(CurrentNode)$;

x **:** add a leaf to $CurrentNode$ labeled with x.

Clearly the Dyck word $\hat{\sigma}$ corresponds to a depth-first search of the fracture tree $FT(\sigma(G))$.

Our goal is to cast this fracture tree (depending on G and on σ) into the modular decomposition tree (depending only on G). It will be shown, by the following properties, that indeed the fracture tree is a good approximation of the modular decomposition tree.

A leaf of the fracture tree of G is identified with the corresponding vertex of x, and a node of the fracture tree is identified with the subset of X rooted by this node, like in a modular decomposition tree.

By definition, the nodes of the decomposition tree represent exactly the strong modules of the graph. For the fracture tree only weaker properties hold, as explained now :

Lemma 2 *Let M be a strong module of G, and G' the graph obtained from G by contracting the module M into a new vertex x. For any factorizing permutation σ of G, $FT(\sigma(G))$ can be obtained from $FT(\sigma[G'])$ by replacing the leaf x by a forest (each connected component of that forest being connected to the father of x in $FT(\sigma[G'])$).*

Proof: Using Lemma 1, vertices of M appear consecutively in σ and therefore $\sigma[G']$ is well-defined, when replacing the vertices of M by a single vertex x.

Let us apply on $\sigma[G']$ the automaton previously introduced. It produces $FT(\sigma[G'])$ in which x is a leaf. Applying the same automaton on σ gives $FT(\sigma(G))$. When the automaton deals with the Dyck word that

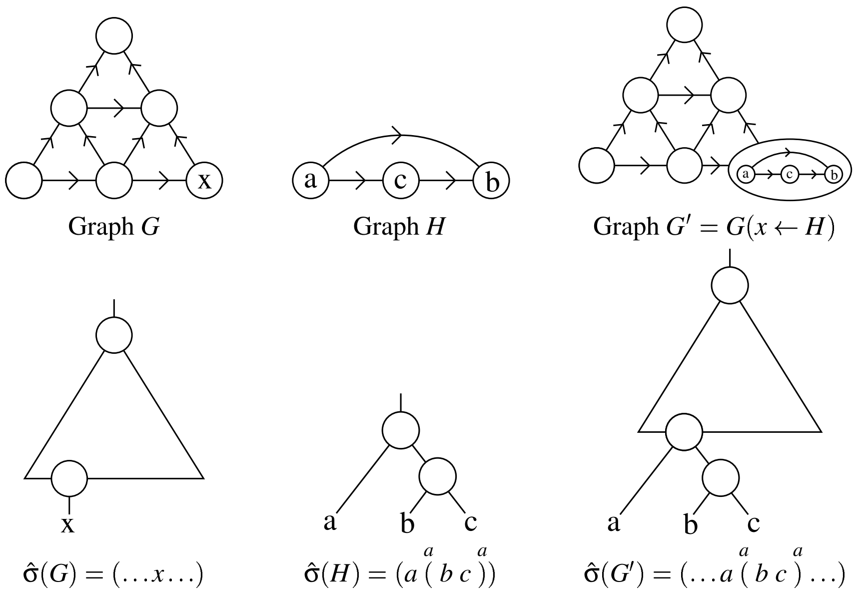


Fig. 5: Substitution in a graph and in a parenthesized factorizing permutation

corresponds to M it produces a forest F. And to obtain $FT(\sigma(G))$ from $FT(\sigma[G'])$ it suffices to replace x by F. □

Therefore we reach here the main technical point of this approach: a strong module does not always correspond to a proper subtree of the fracture tree. It has to be studied in full details, and next theorem describes some good situations.

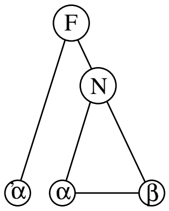
Definition 5 *Given a Dyck word, the* **parentheses height** *at some letter v is the number of opening parentheses before v minus the number of closing parentheses before v.*

Theorem 2 *Let G be a graph, σ a factorizing permutation of G, and N' a node of the modular decomposition tree of G, representing a strong module M. If N' is a prime node, or if the father of N' is a series, parallel or order node, then there is a node N in the fracture tree of σ that represents M.*

Proof: Since the fracture tree is built using the automaton, the vertices of G represented by a node N of the fracture tree are exactly a Dyck subword of $\hat{\sigma}$ enclosed by a pair of parentheses (including the parentheses associated with the link from N to $Father(N)$). Using Lemma 2, it just remains to prove that there is a pair of parentheses enclosing the module, so that its vertices are children of only one node and form a tree. The proof is in two parts: the first deals with series, parallel or order nodes children of series, parallel or order nodes, and the second deals with prime nodes.

Let us start with the easy case. Let N be a parallel, series or order node of the modular decomposition tree and F its parallel, series or order father. First, due to Theorem 1, a series node generates no series children. Similarly a parallel (resp. order) node can not be child of a parallel (resp. order) node. So F and N have different types. Let α be the left end of N in σ and β its right end. Since F has at least two children, the predecessor $'\alpha$ of α or the successor β' of β (or both) exists and descends from F. Suppose $'\alpha$

does. We have $LCA(\alpha,\beta) = N$ and $LCA('\alpha,\alpha) = F$. Let us denote by $|(x,y)|$ the number of arcs between x and y, which can be 0, 1 or 2. By Property 1, $|('\alpha,\beta)| \neq |(\alpha,\beta)|$, so β cuts the pair $\{'\alpha,\alpha\}$. No child z of F to the right of β can cut this pair because its least common ancestor with both $'\alpha$ and α is F so $|(z,'\alpha)| = |(z,\alpha)|$. A vertex out of F can not cut the pair either, because F is a module. β is thus the last cutter of $\{'\alpha,\alpha\}$ and the right fracture of this pair is $\alpha\ldots\beta$. Since N is well parenthesized by itself (by the Lemma 1), this fracture corresponds to a node in the fracture tree. Notice that if $'\alpha$ does not exist or is not a child of F, then β' does, and then N is the left fracture of $\{\beta,\beta'\}$. This demonstrates the theorem in the case of series, parallel or order nodes.



If N is a prime node of the modular decomposition tree, the case is a little more complex. We will prove that whatever the type of his father F, N is a node of the fracture tree. To prove that, we just have to demonstrate that given a prime graph and a parenthesized factorizing permutation of its vertices, the automaton produces a tree and not a forest. The proof uses an induction on the number k of nodes (non leaves) which are children of N.

For $k = 0$ (i.e. all children of N are leaves of the modular decomposition tree), let us suppose that in $\hat{\sigma}[N]$ the parentheses height is null between two consecutive vertices x and x'. $\hat{\sigma}$ can be split into two subwords $\sigma(1)\ldots x$ and $x'\ldots\sigma(h)$. Clearly, xx' can not be a factor of any fracture, because in $\sigma(1)\ldots x$ there is as many opening parentheses as closing ones. Consider the link between x and x'. It can be formed with no arc, one right arc (from x to x'), one left arc or two arcs. Let T be its type. Let us now consider the other links between the two factors. The link between $'x$ (if $'x$ exists) and x' is also T, or else x' would cut the pair $\{'x,x\}$ and xx' would be factor of its right fracture, bringing the parentheses height at more than one: contradiction. By iterating this until the first vertex $\sigma(1)$, all links between a vertex of $\sigma(1)\ldots x$ and x' are also T. For the same reason, all links between x and a vertex of $x'\ldots\sigma(h)$ are T. Now, consider all links between $'x$ (if it exists) and $x'\ldots\sigma(h)$. They are T, because if there exists y to the right of x' such that $('x,y) \neq T$, we know that $(x,y) = T$ so y cuts $\{'x,x\}$ and so $xx' \subset Lf(\{'x,x\})$. A similar argument applies with $''x$ and x'' and so on, hence *all* links between vertices from $\sigma(1)\ldots x$ and from $x'\ldots\sigma(h)$ are T. So $\sigma(1)\ldots x$ and $x'\ldots\sigma(h)$ are two modules. At least one of them is not trivial, because N has at least three vertices, so N can not be a prime graph. In conclusion, the parentheses height can not be null inside the permutation, and therefore $\hat{\sigma}[N]$ is enclosed by two parentheses and N is a node of the fracture tree.

Suppose now that N has $k \geq 1$ nodes for children. Let N' be one of these nodes, and consider the graph G' in which N' is contracted to a single vertex. In the modular decomposition tree of G' the node N has $k-1$ children which are nodes, and therefore we may apply the induction on G'. Therefore N is a node of the fracture tree of G'. Using Lemma 2, we obtain that N is also a node of the fracture tree of G, which demonstrates the theorem for prime nodes. □

Theorem 2 does not cover all cases, in particular: children of a prime node which are series, parallel or order (in the modular decomposition tree) may be not present in the fracture tree. We know by the Lemma 2 that they necessarily appear as consecutive children of one node of the fracture tree. We call

merging such a case. For instance, in our example Figure 4, the order module $\{6,7,8\}$ is merged with the prime module $\{6,\dots 14\}$. And there is no node of the fracture tree representing it.

At this point, we know that there are nodes in the fracture tree representing (almost all) strong modules. Fortunately any vertex of G appears exactly once as a leaf of the fracture tree, and the only leaves of the fracture tree are the vertices of G. So, a node represents the same set of vertices as another if and only if they form a chain of nodes having only one child, excepted the last.

We consider that the true representative of a module is the last node of such a chain, and we say it is a ***genuine*** node. The other nodes (those with only one child, or representing a set of vertices other than a module) are said to be ***dummies***.

Furthermore, it is important to notice that the inclusion of modules (as in the modular decomposition) is preserved in the fracture tree: this is given by Lemma 2. This is why we can claim that the fracture tree is a good approximation of the modular decomposition tree.

So, given the fracture tree, in order to get the modular decomposition tree we have to :

1. identify the genuine nodes and their type,
2. suppress the remaining (dummy) nodes,
3. detect merged strong modules and fix them,
4. suppress the weak modules.

Efficient solutions for all of these four steps are detailed in the next section.

4 Getting the modular decomposition tree

For sake of simplicity let us divide the main algorithm into six successive linear scans. The first one writes the fracture parenthesizing, using the automaton, and the second one reads this parenthesizing and builds the fracture tree. Let us now focus here on the remaining successive scans that transform the fracture tree into the modular decomposition tree.

4.1 Third scan: Module recognition

Obviously M is a module if and only if $Cut(M) = \emptyset$. If S is a factor of σ then we have

$$x \in Cut(S) \Rightarrow \exists\{y,y'\} \subset S \text{ such that } x \in Cut(\{y,y'\})$$

The proof is trivial: if $x \in Cut(S)$ then x cuts some pair $\{u,v\} \in M$. yy' is thus a factor of $u\dots v$. We use the converse: if x does not cut any pair of consecutive vertices of S, then x does not cut S. This leads us to the following property:

Property 4 *Let N be a node of the fracture tree. If all cutters of all consecutive pairs of N belong to N then, N represents a module.*

Let N be the scanned node and $C_1 \dots C_k$ its k children (ordered with respect to σ). Let us call $fv(C_i)$ the first vertex of C_i and $lv(C_i)$ its last vertex, so that C_i is the factor $fv(C_i)\dots lv(C_i)$ of σ. Its cutters can be computed using

$$fc(N) = \min\left(fv(N), \min_{\{x,x'\}\subset N} fc(\{x,x'\})\right)$$

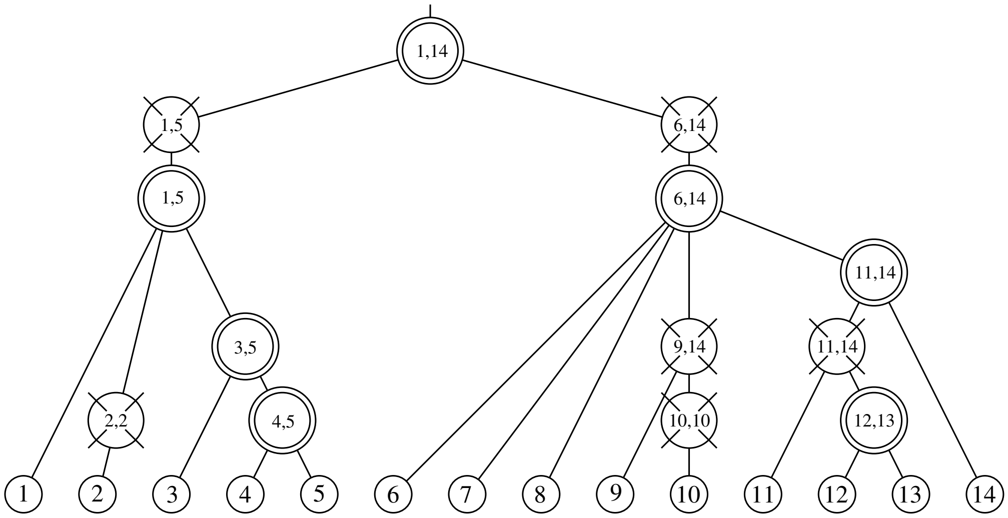


Fig. 6: The fracture tree after the third scan. Genuine nodes are circled, while known dummies are crossed. Inside a node is written its first and last cutters.

$$lc(N) = \max\left(lv(N),\ \max_{\{x,x'\}\subset N} lc(\{x,x'\})\right)$$

where min (resp. max) finds the leftmost (resp. rightmost) vertex in σ.

Notice that the pairs included in a C_i have already been scanned, and in fact there is no need to scan them again. The "new" pairs are only those for which N is the least common ancestor, *i.e.*

$$\{lv(C_i), fv(C_{i+1})\}_{i\in[1,k-1]}.$$

Now we have the following efficient recurrences:

$$fc(N) = \min\Big(fv(N),\ \min_{i=1}^{k-1}\big\{fc\big(\{lv(C_i), fv(C_{i+1})\}\big)\big\}\ ,\ \min_{i=1}^{k}\big\{fc\big(C_i\big)\big\}\Big) \tag{1}$$

$$lc(N) = \max\Big(lv(N),\ \max_{i=1}^{k-1}\big\{lc\big(\{lv(C_i), fv(C_{i+1})\}\big)\big\}\ ,\ \max_{i=1}^{k}\big\{lc\big(C_i\big)\big\}\Big) \tag{2}$$

Property 5 *All the genuine nodes of the fracture tree can be detected in $O(n)$.*

Proof: Suppose we get in $O(1)$ the first and last cutter of a pair (this is realistic, because they are calculated during the parenthesizing scan, and can be stored in an array). With the formulae (1) and (2), a pair may be checked only if the current node is its least common ancestor, and a node only if the current node is its father. So a bottom-up parsing of the fracture tree establishes for any node its first and last cutters and vertices, and then if it is a module, in $O(n)$ for the whole tree. This is the third scan of our algorithm. □

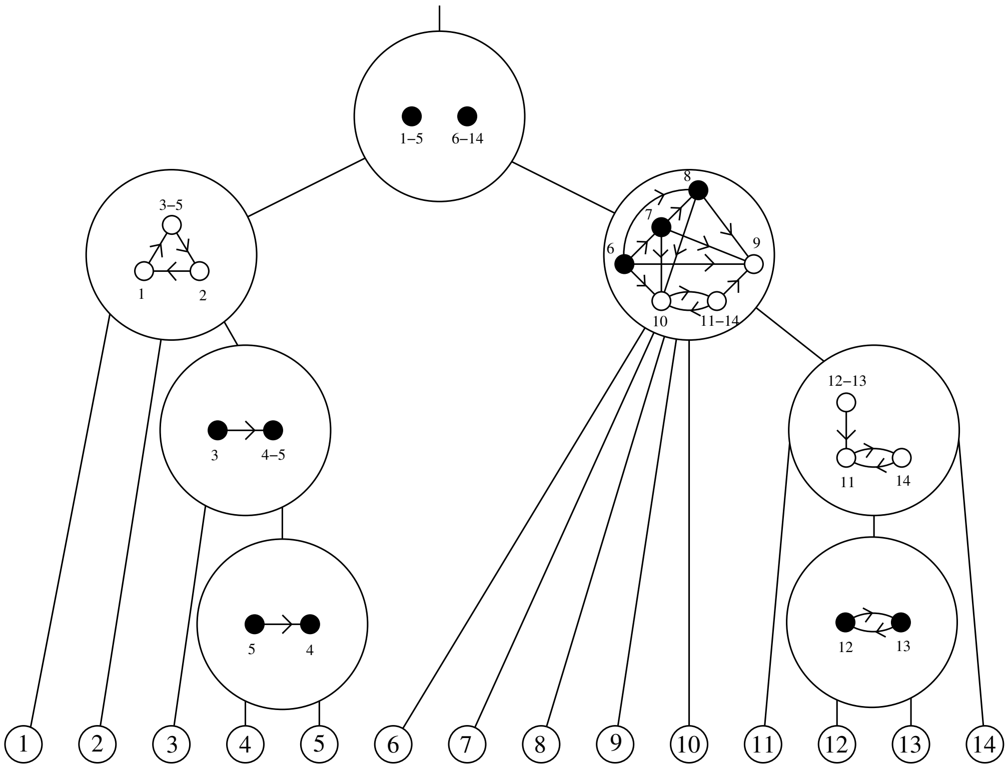


Fig. 7: The fracture tree during the fifth scan. Only genuine nodes remain. Inside a node is written its representative graph, with the twins in black.

4.2 Fourth scan: dummy nodes deletion

Since we know which nodes represent a module and which do not, we can now delete all the dummy nodes (either the nodes with only one child or the nodes that do not represent a module). Notice that by deletion we do not mean that we prune the tree, but that the children of the erased node become children of their former grandfather. The whole process takes $O(n)$, and the new fracture tree produced contains only genuine nodes. It should be noticed also that this step can be very easily merged with the previous one. In fact, up to now among the genuine nodes found in the previous scan, we could have some weak modules, that must be removed from the tree. That is the purpose of the last scan.

4.3 Fifth scan: recovering the merged modules

Given a module M and its children $C_1 \dots C_k$ in the fracture tree, all supposed to be modules at this point, we can focus on its representative graph G_M. Notice that the representative graphs of all the nodes of the tree can be computed in $O(n+m)$. In the following, $\sigma[G_M]$ denotes the factorizing permutation of G_M compatible with $\sigma(G)$, *i.e.* the vertices $x_1 \dots x_k$ of G_M appear in the same order as $C_1 \dots C_k$ in $\sigma(G)$.

If M is an series, order or parallel node, by the Theorem 2 we know that *all* its children in the modular decomposition tree are represented in the fracture tree, and thus are exactly $C_1 \dots C_k$. If M is a series (resp. order, parallel) node then G_M is a clique (resp. total ordering, stable set). This can be checked in $O(k)$: we make an array A_M with the outer degree of each node. If $A_M = [0,\dots 0]$ then M is parallel, and if $A_M = [k-1,\dots k-1]$ then M is series. Obviously in those cases, it is not worthwhile to store G_M and A_M. If A_M is a permutation of $\{0,\dots k-1\}$ and if all vertices of G_M have the same degree (inter plus outer) then M is an order node. The permutation $A_M{}^{-1}$ gives us the total ordering represented by N: for instance $A_M{}^{-1}(0)$ is its smallest element. G_M can be deleted, but $A_M{}^{-1}$ has to be computed and kept.

If none of those cases apply, then M is a prime node. A_M can be deleted, but G_M must be kept, else information about the graph structure will be lost. We must be careful, because the C_i's are not necessary the children of M in the modular decomposition tree: indeed merging of series, parallel or order nodes can happen. Fortunately, since all the children of those merged node are represented, we know that the C_i's are either children (in the modular decomposition tree) of M or of a merged non-prime node. The nodes are ordered according to σ, so that all the children of a merged node appear consecutively. We can detect all merging by searching ***blocks of consecutive twins***.

Definition 6 *Two vertices are twins if no vertex cuts them. x and y are* **true twins** *if $(x,y) \in E$ and $(y,x) \in E$,* **false twins** *if $(x,y) \notin E$ and $(y,x) \notin E$ and* **half twins** *if either $(x,y) \in E$ or $(y,x) \in E$.*

Of course the relation "be a true (resp. false) twin of" is symmetric and transitive. We shall consider the transitive closure of the "be an half twin of" relation, thus we have three equivalences relations.

Theorem 3 *The merged series (resp. parallel, order) nodes are exactly the non-trivial equivalence classes of the twin relations, corresponding to blocks of consecutive true (resp. false,half) twins of maximal length.*

Proof: Let P be a prime node of the modular decomposition tree and M a child of P in this tree, that merged with P in the fracture tree. Since M is a strong module, it is a factor $x_l \dots x_r$ $(l < r)$ of the factorizing permutation $\sigma[G_P]$. If M is series or parallel, it is obvious that

$$\forall i,j \in [l,r],\ x_i \ and\ x_j\ are\ twins\ (true\ or\ false) \tag{3}$$

In fact (3) holds also if M is an order module. Let us prove it. Suppose there are two vertices x_i and x_j of $x_l \dots x_r$ which are not twins. Necessarily there exist two consecutive non-twins vertices x_k and x'_k between x_i and x_j. Since M is a module, the cutters of $\{x_k, x'_k\}$ belong to M. So there is a fracture containing some (at least two, but not all) consecutive vertices of $x_l \dots x_r$. The automaton that has produced the fracture tree would have created one node for this fracture, a contradiction. Thus all consecutive vertices of a merged node are twins in the representative graph G_P. A vertex x of a merged module M can not be a twin of $y \notin M$, so merged modules are indeed equivalence classes of the transitive closure of the twin relation. Since those classes are by construction factors of $\sigma(G_P)$, this demonstrates the theorem. □

It should be noticed that in the merged order case, the ordering of the vertices in σ is exactly the ordering induced by the order node, otherwise there would exist some cutter.

We know without further calculation the cutters of a pair: if $fc(\{lv(C_i), fv(C_{i+1})\}) \in C_j,\ j < i$ in G, then $fc(\{x_i, x'_i\}) = x_j$ in G_M, and if $fc(\{lv(C_i), fv(C_{i+1})\}) \in C_i$ in G then $\{x_i, x'_i\}$ has no first cutter in G_M. No other case is possible. Let us now consider how twins can be detected: x_i and x'_i are twins if and only if the first (resp. last) cutter of $\{lv(C_i), fv(C_{i+1})\}$ does not exist or is to the right (resp. left) of $fv(C_i)$ (resp. $lv(C_{i+1})$). Thus a pair $\{x, x'\}$ of $\sigma(G)$ is used only once in the search of twins, in the node $LCA(x,x')$, making this search $O(n)$ for the whole tree.

4.4 Last scan: weak module deletion

At this point we know that *all* the strong modules are represented in our tree (which is now very near to the modular decomposition tree), and that *all* its nodes are modules. It only remains to determine the genuine nodes which do not correspond to strong modules of the original graph. Since a node of the fracture tree always represents a factor of σ, the weak modules factor of $\sigma(G)$ can only be factors of series, parallel or order nodes. And we know that those kind of nodes can not have children with the same type as them. Thus we detect a weak module node because we see a series node being a child of a series node (it is the same for parallel or order nodes). They were the last dummies, and they are deleted from the tree: the tree we have now is the modular decomposition tree. In our example, the order module $\{4,5\}$ is weak and must be merged with the (strong) module $\{3,4,5\}$.

In fact, the only weak module nodes are the orders. In a series or parallel module, all the vertices are twin in the representative graph, so its parenthesized factorizing permutation has in fact no parentheses.

5 Complexity

Now we can state our main result:

Theorem 4 *Our algorithm computes the modular decomposition tree of any directed graph, if a factorizing permutation is provided, in $O(n+m)$ time and space complexity.*

Proof:

1. The first scan, which establishes the parenthesizing of a given factorizing permutation, takes $O(n+m)$ time, because for the $n-1$ pairs of vertices we search the first and last cutter in the adjacency of their two vertices, so the adjacency of a vertex is examined at most twice. The output of this scan is a permutation with n vertices and at most $4n-4$ parentheses,

2. The second scan, which reads the $O(n)$-sized parenthesizing and builds the fracture tree with at most $2n-2$ nodes, takes $O(n)$ space and time since it uses a simple automaton,

3. The third scan (modules recognition) takes $O(n)$ time complexity (Property 5),

4. The fourth scan (dummy nodes deletion) takes $O(n)$ time, since it uses a breadth-first search, and suppress in $O(1)$ (by an easy merging of the lists of children) the dummy nodes,

5. The fifth scan takes $O(n+m)$ time and space complexity for quotienting all subgraphs. Then, the module typing and the search of twins (knowing the cutters) is $O(n)$,

6. The sixth and last scan takes $O(n)$ for deleting weak modules.

Thus the whole complexity of the proposed algorithm is in $O(n+m)$. □

6 Conclusions

A consequence of our main result is that the modular decomposition of graphs and the computation of a factorizing permutation with respect to the strong modules are really equivalent problems. It was know that these two problems have the same theoretical time complexity. But here we have proposed an algorithm which is not only theoretically interesting but really efficient.

This allows some new perspectives about using a systematic search of factorizing permutations in many graph decomposition contexts. See for example the decomposition of inheritance graphs as used in Object Oriented Programming and Knowledge Representation Systems. The block decomposition of these graphs is computed using a two phases algorithm: the first one computes a factorizing permutation, and the second one extracts the strong blocks from it [2].

We have pointed out that the factorizing permutations play a central role in the decomposition theory. Another direction worthwhile investigating is the question of which invariants can be directly computed out of factorizing permutations. It remains also to consider new classes of graphs for which a factorizing permutation is easy to compute. Generalizations of chordal graphs (HHD-free graphs for example) are natural candidates.

And we may end with a nice problem, which is a enlightened by this work and some recent linear modular decomposition algorithms [4, 20, 7], which consists in proving the following conjecture.

Conjecture:

There exists a simple way to compute a factorizing permutation for the modular decomposition of any undirected graph.

Up to now only an $O(n+m\log n)$ direct algorithm is known to compute a factorizing permutation for any undirected graphs [13]. For cographs it has been recently improved down to linear-time [12].

Another direction of research could be to generalize the notion of factorization permutation to other decompositions such as the split decomposition of graphs [6, 18]. This would be very interesting because in such a decomposition the decomposition tree is quite different.

Acknowledgements

The authors wish to thank the referees for their valuables remarks and suggestions.

References

[1] C. Capelle. Block decomposition of inheritance hierarchies. In R.H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science - WG'97*, number 1335 in LNCS, pages 118–131, Berlin, June 1997.

[2] C. Capelle. *Décompositions de Graphes et Permutations Factorisantes*. PhD thesis, Université Montpellier II, January 1997.

[3] C. Capelle and M. Habib. Graph Decompositions and Factorizing Permutations. In *proceedings of ISTCS'97*, pages 132–143, Ramat Gan (Israel), June 1997. IEEE.

[4] A. Cournier and M. Habib. A new linear algorithm for Modular Decomposition. In S. Tison, editor, *Lectures notes in Computer Science, 787. Trees in Algebra and Programming-CAAP'94*, pages 68–84. Springer-Verlag, April 1994. 19th International Colloquium, Edinburgh, U.K., April 1994. Procedings.

[5] W. H. Cunningham and J. Edmonds. A combinatorial decomposition theory. *Canad. J. Math*, 32(3):734–765, 1980.

[6] W.H. Cunningham. Decomposition of directed graphs. *SIAM J. Algebraic Discrete Methods*, 3:214–228, 1982.

[7] E. Dahlhaus, J. Gustedt, and R. M. McConnell. Efficient and Practical Modular Decomposition. In *8th Annual ACM-SIAM Symposium On Discrete Algorithms (SODA)*, pages 26–35, January 1997.

[8] R. Ducournau and M. Habib. La multiplicité de l'héritage dans les langages à objets. *Technique et Science Informatique*, 8(1):41–62, 1989.

[9] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, New-York, 1980.

[10] M. Habib, M. Huchard, and J. Spinrad. A linear algorithm to decompose inheritance graphs into modules. *Algorithmica*, 13:573–591, 1995.

[11] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutives ones testing. *Theoretical Computer Science*, 234:59–84, 2000.

[12] M. Habib and C. Paul. A simple linear time algorithm for cograph recognition. submited 2000.

[13] M. Habib, C. Paul, and L. Viennot. Partition Refinement techniques: an interesting toolkit. *International Journal of Foundations of Computer Science*, 10(2):147–170, 1999.

[14] J. E. Hopcroft and R. E. Tarjan. Dividing a Graph into Triconnected Components. *SIAM Journal of Computing*, 2(3):135–158, 1973.

[15] Wen-Lian Hsu. A Simple Test for Interval Graphs. *Lecture Notes in Computer Science*, 657:11–16, 1992.

[16] Wen-Lian Hsu and Tze-Heng Ma. Substitution decomposition on chordal graphs and applications. In *Proceedings of the 2nd ACM-SIGSAM International Symposium on Symbolic and Algebraic Computation*, pages 52–60, 1991.

[17] M. Huchard. *Sur quelques questions algorithmiques de l'h éritage multiple*. PhD thesis, Université Montpellier II, 1992.

[18] T. Ma and J. Spinrad. An $O(n^2)$ algorithm for undirected split decomposition. *J. of Algorithms*, 16(1):145–160, 1994.

[19] R. McConnell and J. Spinrad. Linear-Time Modular Decomposition and Efficient Transitive Orientation of Undirected Graphs. In *Proc. of the fifth Annual ACM-SIAM Symposium of Discrete Algorithms*, pages 536–545, 1994.

[20] R. M. McConnell and J. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201:189–241, 1999.

[21] R. H. Möhring and F. J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Ann. Discrete math*, 19:257–356, 1984.

[22] C. Paul. *Parcours en Largeur Lexicographique : Un Algorithme de Partitionnement, Application aux Graphes et Généralisations*. PhD thesis, Université Montpellier II, January 1998.